НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
“___” червня 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки **6.050102 «Комп'ютерна інженерія»**

на тему: Алгоритм і програма формування діагностичних графів багатопроцесорних систем

Виконала: студентка IV курсу, групи КВ-52
Коваленко Олена Павліна _____
(підпис)

Керівник: професор кафедри СПіСКС, д.т.н., доцент
Романкевич В. О. _____
(підпис)

Консультант з нормоконтролю, доц.каф.СПіСКС, к.т.н.
Клятченко Я.М. _____
(підпис)

Рецензент _____
(підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2019 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4	ІАЛЦ. 045490.000	Завдання на дипломний проект	2	
2	А4	ІАЛЦ. 045490.001 ОА	Опис альбому	2	
3	А4	ІАЛЦ. 045490.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ. 045490.003 ТП	Відомість технічного проекту	2	
5	А4	ІАЛЦ. 045490.004 ПЗ	Пояснювальна записка	53	
6	А4	ІАЛЦ. 045490.005 Д1	Взаємодія модулів програми. Схема структурна	1	
7	А4	ІАЛЦ. 045490.006 Д2	Алгоритм формування ваги для кожного ребра. Схема алгоритму	1	
8	А4	ІАЛЦ. 045490.007 Д3	Побудова діагностичного графа. Схема алгоритму	1	
9	А4	ІАЛЦ. 045490.008 Д4	Формування “0”-ланцюжків. Схема алгоритму	1	
10		CD-ROM	Матеріали дипломного проекту		

				ІАЛЦ. 045490.000		
	ПІБ	Підп.	Дата			
Розробн.	Коваленко О.П.			Відомість дипломного проекту	Лист	Листів
Керівн.	Романкевич В.О.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. СПіСКС Гр. КВ-52	
Н/контр.	Клятченко Я.М.					
Зав.каф.	Тарасенко В.П.					

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
«___» червня 2019 р.

ЗАВДАННЯ
на дипломний проект студентки
Коваленко Олени Павлівни

1. Тема проекту “Алгоритм і програма формування діагностичних графів багатопроцесорних систем”, керівник проекту доцент кафедри СП і СКС, д-р техн. наук Романкевич В.О. , затверджені наказом по університету від «22» травня 2019 р. №1330-С

2. Термін подання студентом проекту 11 червня 2019 р.

3. Вихідні дані до проекту: див. Технічне завдання.

4. Зміст пояснювальної записки:

- аналіз існуючих рішень та обгрунтування теми дипломного проекту;
- обгрунтування вибору засобів реалізації;
- програма візуалізації графів-циркулянтів.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо):

- Взаємодія модулів програми. Структура.
- Формування ваги для кожного ребра. Схема алгоритму.
- Побудова діагностичного графа. Схема алгоритму.
- Формування “0”-ланцюжків. Схема алгоритму.

- Презентація дипломного проекту.

6. Консультанти розділів проекту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	доц.каф.СПіСКС, к.т.н. Клятченко Я.М.		

7. Дата видачі завдання 07.11.2019

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	15.11.2018	
2.	Розроблення та узгодження технічного завдання	30.11.2018	
3.	Аналіз існуючих рішень	05.02.2019	
4.	Підготовка матеріалів першого розділу дипломного проекту	05.05.2019	
5.	Підготовка матеріалів другого розділу дипломного проекту	07.05.2019	
6.	Підготовка матеріалів третього розділу дипломного проекту	10.05.2019	
7.	Підготовка матеріалів четвертого розділу дипломного проекту	16.05.2019	
8.	Підготовка графічної частини дипломного проекту	20.05.2019	
9.	Оформлення документації дипломного проекту	24.05.2019	
10.	Попередній огляд матеріалів диплому на кафедрі	29.05.2019	

Студент _____ Коваленко О.П.
(підпис)

Керівник проекту _____ Романкевич В.О.
(підпис)

* Консультантом не може бути зазначено керівника дипломного проекту.

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (53 с., 20 рис., 3 додатка).

Об’єкт розробки – створення програми візуалізації графів-циркулянтів, що надає можливість ілюструвати процес тестування у багатопроцесорних системах.

В процесі розробки була використана мова програмування Java з використанням бібліотек Swing - для розроблення графічного інтерфейсу та JUNG - для побудови елементів графів. Було обрано середовище розробки Eclipse.

Розроблена програма дозволяє:

- задання вхідних даних користувачем;
- візуальне представлення графа-циркулянта згідно з вхідними даними;
- можливість редагування, перетягування вершин і масштабування;
- збереження зображення графа у форматі png.

В ході виконання дипломного проекту:

- розроблено архітектуру програми;
- проведено аналіз існуючих рішень;
- досліджено засоби реалізації;
- розроблено алгоритм побудови циклічних графів.

Ключові слова: візуалізація, граф, багатопроцесорні системи, цикл, вершина, ребро, JAVA, JUNG, Swing.

ABSTRACT

Qualifying work includes an explanatory note (53 p., 20 fig., 3 applications).

The object of development is creating a circle graph visualization program that provides an opportunity to illustrate the testing process in multiprocessor systems.

During the development were used programming language Java using framework Swing - for developing a graphical interface and JUNG - for building graphs elements. Eclipse was chosen as the development environment.

The program allows you to:

- provision of initial data by the user;
- visual representation of the circular graph according to the input data;
- the ability to edit, zoom and drag the vertices;
- possibility to save graphic image in png format.

During the development of the diploma project:

- the architecture of the system was created;
- the analysis of existing solutions was made;
- the means of realization were investigated;
- the algorithm for constructing cyclic graphs was developed.

Keywords: visualization, graph, multiprocessor systems, circle, vertex, edge, JAVA, JUNG, Swing.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ. 045490.002 ТЗ	Алгоритм і програма	4		
			формування діагностичних			
			графів багатопроцесорних			
			систем.			
			Технічне завдання			
	А4	ІАЛЦ. 045490.003 ТП	Алгоритм і програма	2		
			формування діагностичних			
			графів багатопроцесорних			
			систем.			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ. 045490.004 ПЗ	Алгоритм і програма	53		
			формування діагностичних			
			графів багатопроцесорних			
			систем.			
			Пояснювальна записка			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ. 045490.001 ОА		
Розробив		Коваленко О.П.			Алгоритм і програма формування діагностичних графів. Опис альбому	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Романкевич В.О.					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-52	
Затвердив		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ. 045490.005 Д1	Взаємодія модулів	1		
			програми.			
			Схема структурна			
	А4	ІАЛЦ. 045490.006 Д2	Алгоритм формування ваги	1		
			Для кожного ребра.			
			Схема алгоритму			
	А4	ІАЛЦ. 045490.007 Д3	Побудова діагностичного	1		
			графа.			
			Схема алгоритму			
	А4	ІАЛЦ. 045490.008 Д4	Формування	1		
			“0”-ланцюжків.			
			Схема алгоритму			
		CD-ROM	Матеріали дипломного			
			проекту			

Зміст

					ІАЛЦ.045490.002 ТЗ		
Зм.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Коваленко О.П.			Алгоритм і програма формування діагностичних графів багатопроцесорних систем.	Літ.	Аркуш 1 / Аркушів 4
Перевір.		Романкевич В.О.					
Н. контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-52	
Затв.		Тарасенко В.П.					

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування роботи – «Алгоритм і програма формування діагностичних графів багатопроцесорних систем».

Область застосування: діагностика у багатопроцесорних системах.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання дипломного проекту першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення програми візуалізації графів, що надає можливість ілюструвати процес тестування у багатопроцесорних системах.

4. ДЖЕРЕЛА РОБОТИ

Джерелами інформації для розроблення є технічна література, публікації у періодичних виданнях та Інтернет ресурси з питань розробки.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до системи, що розробляється

Система повинна забезпечувати наступні функції:

- Задання вхідних даних користувачем.
- Візуальне представлення графа згідно з вхідними даними.
- Можливість редагування, перетягування вершин і масштабування.
- Можливість збереження зображення графа у зручному для користувача форматі.

5.2. Рекомендовані вимоги до апаратного забезпечення

- Процесор: Intel® Pentium®.
- Оперативна пам’ять: 4 Гб.
- Простір на диску: 2 Гб.
- Тип системи: 64-розрядна операційна система.

5.3. Вимоги до мінімального програмного забезпечення

- Операційна система Windows, Linux чи Mac OS X.
- Комплект розробника застосунків Java Development Kit 8.
- Бібліотека JUNG 2.0.1.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.11.2018
2.	Розроблення та узгодження технічного завдання	30.11.2018
3.	Аналіз існуючих рішень	05.02.2019
4.	Підготовка матеріалів першого розділу дипломного проекту	05.05.2019
5.	Підготовка матеріалів другого розділу дипломного проекту	07.05.2019
6.	Підготовка матеріалів третього розділу дипломного проекту	10.05.2019
7.	Підготовка матеріалів четвертого розділу дипломного проекту	16.05.2019
8.	Підготовка графічної частини дипломного проекту	20.05.2019
9.	Оформлення документації дипломного проекту	24.05.2019
10.	Попередній огляд матеріалів диплому на кафедрі	29.05.2019

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ. 045490.002 ТЗ	Завдання на дипломний	2		
			проект			
	А4	ІАЛЦ. 045490.001 ОА	Опис альбому	2		
	А4	ІАЛЦ. 045490.002 ТЗ	Технічне завдання	4		
	А4	ІАЛЦ. 045490.003 ТП	Відомість технічного	2		
			проекту			
	А4	ІАЛЦ. 045490.004 ПЗ	Пояснювальна записка	53		
	А4	ІАЛЦ. 045490.005 Д1	Взаємодія модулів	1		
			програми.			
			Схема структурна			
	А4	ІАЛЦ. 045490.006 Д2	Алгоритм формування ваги	1		
			Для кожного ребра.			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ. 045490.003 ТП		
Розробив		Коваленко О.П.			Алгоритм і програма формування діагностичних графів. Відомість технічного проекту	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Романкевич В.О.					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-52	
Затвердив		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ. 045490.007 Д3	Побудова діагностичного	1		
			графа.			
			Схема алгоритму			
	А4	ІАЛЦ. 045490.008 Д4	Формування	1		
			“0”-ланцюжків.			
			Схема алгоритму			
		CD-ROM	Матеріали дипломного			
			проекту			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ. 045490.003 ТП	Арк. 2

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API (Application Programming Interface) – це набір готових класів, процедур, функцій, структур і констант, що надаються додатком (бібліотекою, сервісом);

AWT (Abstract Window Toolkit) – бібліотека графічного інтерфейса;

DGML(Directed Graph Markup Language) – файловий формат для опису графів;

DOT (Graph description language) – мова описання графів;

GEA (Graph Extended Algol) – мова обробки графів;

GML (Game Maker Language) – інтерпретована мова програмування;

GraphML (Graph Markup Language) – мова описання графів на основі XML; файловий формат для опису графів;

Graphviz – пакет утиліт для автоматичної побудови графів;

GRASPE – мова програмування, використовувалася для опису графів;

GUI (Graphical user interface) – графічний інтерфейс користувача;

GXL (Graph eXchange Language) – мова для зображення графів;

JDK (Java Development Kit) – комплект для розробників мови Java;

JGraph – це бібліотека для малювання графів;

JVM (Java Virtual Machine) – віртуальна машина Java;

JUNG – це бібліотека для візуалізації графів;

JVM (Java Virtual Machine) – віртуальна машина Java;

LISP – мова програмування з підтримкою парадигм функціонального та процедурного програмування;

PDF – текстовий формат;

Postscript – це мова програмування для опису вміст сторінки;

SVG (Scalable Vector Graphics) – мова для розмітки графіків;

Swing – бібліотека для створення GUI в Java;

TGF (Trivial Graph Format) форма файлів, оснований на тексті, для опису графів;

Wolfram Mathematica – система для складних математичних обчислень;

YEd – це багатофункціональний редактор для ієрархічних діаграм та блок–схем ;

БС – багатопроцесорна система;

ЕОМ – електронно–обчислювальна машина;

ОС – операційна система.

ВСТУП

Поняття графа є одним з фундаментальних в таких науках, як математика та інформатика. Велика кількість теоретичних і практичних задач зводиться до використання алгоритмів на графах. На сьогодні існує достатньо багато спеціалізованих бібліотек для всіх популярних мов програмування, в яких реалізовано фундаментальні алгоритми та складові візуального представлення графа. Однак в більшості уже існуючих рішень відображення графа виконано для загального випадку, не враховуючи особливості окремих графів.

В сфері програмування існує багато бібліотек чи програм для візуалізації графів. Для кожного окремого класу графа потрібно шукати своє рішення для більш зрозумілого зображення та забезпечення необхідних умов. Не існує універсальної програми для представлення графів, тому для кожної окремої задачі необхідно знаходити найбільш правильне рішення.

По направленню, що пов’язане з дослідженням алгоритмів і методів візуалізації графів, за яким в світовій практиці закріпилась назва “Graph Drawing”, щорічно проводяться міжнародні конференції, перша з яких відбулася в 1994 році. Матеріали цих конференцій регулярно публікуються в серії книг Lecture Notes in Computer Science, а огляд алгоритмів малювання графів містить більш ніж 300 робіт, що були написані ще задовго до першої конференції по даній тематиці.

Останнім часом теорія графів стала потужним засобом дослідження і вирішення багатьох завдань, що виникають при вирішенні великих, складних систем.

Дійсно, існує ряд систем, вивчення яких стає значно простіше з використанням теорії графів. Крім того теорія графів виявилася корисною при вивченні завдань, що виникають в деяких областях наук, таких як: теорії груп, теорії матриць, теорії інформації.

В даний час, з розвитком науково–технічного прогресу, вивчення теорії графів знайшло актуальність у зв'язку із їх застосуванням при дослідження роботи багатопроцесорних систем (БС).

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

Вирішенням задач візуалізації займається сфера комп’терних наук і математики, а саме теорія графів із засобами відображення даних. Завдяки стрімкому розвитку мережі Інтернет на сьогоднішній день теорія графів використовується для аналізу даних у різноманітних сферах, таких як соціальні мережі, картографія, лінгвістика та біоінформатика. Спосіб графічного представлення графа залежить від завдання, що треба виконати. Щоб відобразити граф, достатньо знати кількість вершин та взаємозв’язки між ними, що представляють собою ребра.

Потрібно розрізняти поняття зображення графа та графа, як структури даних, оскільки одній структурі графа може відповідати більш ніж одне графічне представлення. Головна задача зображення – показати зв’язки і напрямок між вершинами. Часто на практиці важко відповісти на питання , чи є два зображення моделями одного й того ж графа. В залежності від задачі, одні зображення можуть давати більш наглядну картину, ніж інші. Саме тому рішення проблеми, що вже існують, не завжди в достатній мірі підходять для конкретної задачі.

Існуючі системи візуалізації графів можна умовно розділити на два класи.

До першого класу відносяться вузькоспеціалізовані системи, які орієнтовані на моделі графів з певною семантикою та топологією. Кожна така система є, як правило, частиною деякого більшого проекту, виконуючи роль візуалізатора специфічних для даної роботи даних. Більш широке використання таких систем або просто неможливе, або значно ускладнює процес.

Другий клас – це універсальні системи візуальної обробки графічних моделей, такі як, наприклад, системи daVinci, GraphEd, Graphlet, GraVis, VCG,

а також бібліотеки LEDA, AGD, ffGraph, Graph Layout Toolkit, Graph Editor Toolkit. Не дивлячись на значний прогрес в створенні універсальних систем, варто відмітити ряд недоліків більшості з них. Насамперед для користувача, що звик до операційної системи (ОС) Windows, недоліком цих систем буде їх орієнтація на роботу в ОС UNIX, на великих робочих станціях, якими в достатній мірі оснащені іноземні університети, що їх розробляють. Як правило, універсальність існуючих систем досить обмежена[1].

1.1. Мови програмування для зображення графів

Роботи по розробці мов програмування, що оперують поняттями теорії графів, ведуться з 70х років минулого століття. Кожна з розроблених мов використовувалася для вирішення прикладних завдань і була розширенням мови програмування, що найбільш широко використовувалася на той час.

На сьогодні є досить велика кількість розроблених мов програмування, що використовують теорію графів. Кожна з таких мов знаходила застосування при вирішенні прикладних задач і була доповненням до популярних на той час мов програмування.

Було проаналізовано засоби для візуального представлення графів[2,3]:

1) Однією з перших мов програмування була Graph Extended Algol (GEA), яка була створена у 1970 році. Перевагою цього засобу була простота реалізації, адже для вирішення завдання обробки графів за допомогою GEA достатньо мати базові навички програмування. Найбільший недолік цієї розробки є непродуманість прикладного застосування, тому на сьогоднішній день GEA є застарілою.

2) DOT (Graph description language). DOT дозволяє змінювати зовнішній вигляд графа. До параметрів, які можна змінювати, відносяться колір і шрифт текста, стиль стрілок та рамок і т. п. Можна використовувати атрибут «URL», задаючи відносні або абсолютні гіперпосилання для вузлів і ребер. Також є кілька методів для автоматичного малювання неорієнтованих

графів. Істотним недоліком є відсутність автономності цього засобу розробки, адже граф представлений у текстовому вигляді і для його візуалізації необхідно використовувати ще один додаток. Так як DOT розподіляє елементи автоматично оптимальним чином, а також не передбачається встановлення позицій. У зв'язку з цим елементи зображуються не так, як очікується. Саме тому доводиться редагувати вручну розташування графа.

3) GraphML(Graph Markup Language). GraphML – це файловий формат, що використовується для опису графів. Даний засіб розробки дозволяє описати орієнтовані та неорієнтовані, змішані та ієрархічні графи, а також гіперграфи. На базі GraphML було створено систему Visual Graph, яка підтримує обробку довільних ієрархічних графів, в тому числі і кластерних. Основними задачами Visual Graph є наглядне відображення уже існуючих графових моделей і зручна навігація для них, але без можливості редагування, в чому і заключається головний недолік системи.

4) Trivial Graph Format (TGF) надає функціонал для опису безпосередньо структури графа, з можливістю призначати вершинам і ребрам по одному найменуванню. У графічному редакторі yEd і пакеті Wolfram Mathematica підтримується формат TGF. До недоліків можна віднести те, що не передбачено можливість вказувати додаткові положення, позицію в просторі і змінювати зовнішній вигляд елементів. Також TGF не забезпечує підтримку вкладених структур графа.

5) Game Maker Language(GML) – один із головних попередників GraphML. GML – це інтерпретована мова програмування, розроблена для використання разом з програмою для розробки комп'ютерних ігор, що називалася Game Maker. GML був одним із основних файлових форматів для Graphlet, а також підтримується рядом інших систем обробки графів. Він є менш розвинутим в частині підтримки атрибутів та в частині підключення інформації для використання синтаксичних аналізаторів. Це

вузькоспеціалізований засіб, який не забезпечить необхідних можливостей для вирішення завдання дипломного проекту.

6) Мова Graph eXchange Language (GXL) є спробою розширити мову TXL, що використовується для дерев. Від TXL засіб успадкував синтаксис, мовні особливості і стиль, адаптувавши і розширивши їх для роботи зі спрямованими графами. Серед важливих рис TXL, перенесених в GXL, варто відзначити строгу типізацію, принцип локальності дії правил (можливість обмеження дії трансформації на деякий підграф або піддерево, а не на весь граф) і параметризацію (можливість роботи з декількома незалежними копіями підграфа або піддерева і передачі їх в якості параметрів). Мові GXL властивий загальний недолік всіх систем переписування – необхідність піклуватися про завершеність процесу переписування (процес може привести до зациклення). Крім того, відчувається недостатня гнучкість мови: опис досить простих трансформацій вимагатиме значних зусиль для реалізації цього в рамках даної моделі.

Directed Graph Markup Language (DGML) – XML–подібний формат файлів для орієнтованих графів. Використовується для опису і подальшої візуалізації відношень і залежностей в програмному коді. Мова DGML описує інформацію, яка використовується для візуалізації. Для опису циклічних та ациклічних спрямованих графів в DGML використовується простий XML–код. Спрямований граф представляє собою набір вузлів, що з’єднані між собою посиланнями або границями. Вузли і посилання можуть бути використані для представлення мережевих структур, наприклад елементів програмного проекта. Основним недоліком є вузька спеціалізація.

1.2. Бібліотеки для візуалізації графів

В даному розділі було проаналізовано програмне забезпечення для візуалізації графів. Існують такі популярні бібліотеки візуалізації графів, як Graphviz, JGraph і JUNG[4].

Graphviz є програмним забезпеченням з відкритим вихідним кодом. Graphviz дозволяє створювати граф на основі його текстового опису. Основними особливостями даного продукту є те, що отримані з його допомогою графи можна експортувати в різні формати, наприклад, SVG для веб–сторінок, PDF або Postscript для використання в інших документах. Також дане програмне забезпечення дозволяє відображати графи в браузерах і має безліч різних налаштувань, таких як вибір варіантів кольору, шрифтів, стилів зв'язків між вершинами графа.

JGraph є фреймворком з відкритим вихідним кодом, доступним для мов програмування Java, C # і JavaScript. Спочатку, JGraph був створений як архітектурне розширення класу JTree для мови програмування Java. Клас JTree є компонентом бібліотеки Swing для відображення дерев. JGraph надає можливості редагування графів, але семантика графів, тобто число вершин і зв'язків, визначається додатками, що використовують JGraph. Отже, JGraph надає гнучкі можливості зміни графів, у випадках, коли цього недостатньо, програміст може змінити стандартну реалізацію вихідного коду використовуваних компонент бібліотеки JGraph.

JUNG є фреймворком візуалізації графів з відкритим вихідним кодом, написаним на мові програмування Java. JUNG надає безліч вбудованих алгоритмів представлення графа, а також алгоритмів аналізу. JUNG має здатність підтримувати різні способи представлення графів, наприклад, такі структури даних, як орієнтовані і неорієнтовані графи. JUNG також забезпечує механізми анотації графів, вершин і зв'язків між вершинами. Фреймворк полегшує створення аналітичних інструментів для складних наборів даних, які можуть аналізувати відносини між вершинами. JUNG включає в себе реалізацію цілого ряду алгоритмів з теорії графів, аналізу даних і аналізу соціальних мереж, такі як кластеризація, декомпозиція, оптимізація, генерація випадкових графів, статистичного аналізу та розрахунку відстані між вершинами[5].

Окрім вище згаданих інструментів для візуалізації графів, також варто взяти до уваги й менш відомий UCINET. UCINET переважно використовується серед дослідників соціальних мереж для виконання стандартних методів аналізу соціальних мереж для графів. Однак UCINET не може бути вбудований в додатки: відсутня функція викликати UCINET на дисплеї кінцевого користувача, а JUNG надає засоби для динамічного зміни графів, програмного виклику коду і виведення результатів у міру продовження програми[6].

На підставі проведеного аналізу програмного забезпечення, було прийнято рішення використовувати фреймворк JUNG на базі мови програмування Java. Його основними перевагами є можливість використовувати JUNG для створення власного подання орієнтованого графа. Також JUNG без проблем взаємодіє з графічним інтерфейсом GUI (Graphical user interface) Swing, за допомогою якого створюються компоненти та оброблюються події.

1.3. Обгрунтування теми дипломного проекту

Метою даного дипломного ранку є побудова графів для ілюстрації процесу тестування у БС.

За останні роки сфера використання БС значно розширилася. Поява БС була викликана потребою вирішення складних задач з досить великим об'ємом обчислень та обмеженістю максимальної швидкодії класичних електронних обчислювальних машин (ЕОМ). Незалежно від характеристик комп’ютерного обладнання, продуктивність ЕОМ збільшується при використанні БС.

З розвитком обчислювальної техніки вимоги до швидкодії ЕОМ також зростають. Час перемикання між електронними схемами досягнув долі наносекунди, в той же час швидкість поширення сигналів в лініях, що пов'язують вузли і елементи машини, обмежена швидкістю світла (30

см/нс)[7]. Саме тому істотного підвищення продуктивності ЕОМ не відбудеться зі зменшення часу перемикання електронних схем. При таких умовах підвищення швидкодії ЕОМ можливе при використанні принципу паралелізму для пристроїв обробки даних та створенні багатомашинних та багатопроцесорних систем.

Багатопроцесорні системи сьогодні використовуються повсюдно і мають високі теоретичні показники. Однак при вирішенні більшості практичних завдань при відмові роботи одного з процесорів, необхідно, щоб система працювала, незважаючи на заміну процесора, що вийшов з ладу. Це реальна проблема, яка актуальна для більшості БС, що існують на сьогоднішній день.

БС стануть основною технічною базою обчислювальних центрів всіх рівнів на найближчі роки. В області периферійних пристроїв планується значне розширення їх номенклатури і підвищення їх якісних показників.

Найважливішою складовою частиною створення і впровадження високопродуктивних технологій є істотне розширення комп'ютерної освіти і підготовка кваліфікованих фахівців у цій галузі за останні роки. У багатьох вузах проводиться підготовка фахівців з даного напрямку.

Як показують дослідження, для спрощення основних обчислень при вирішенні переважної більшості задач обробки сигналів найефективніше використовувати функціональні можливості матричних операцій. Були проведені дослідження в сфері використання різноманітних обчислювальних методів лінійної алгебри, в результаті яких було розроблено стійкі пакети програм, за допомогою яких є можливість виконання цих операцій для комп’ютерів послідовної дії. Збільшивши порядок швидкості обчислень, виникає можливість виконати велику кількість алгоритмів. Не дивлячись на успіхи в сфері цифрових інтегральних схем, неможливо повністю покладати надії на досягнення технологій цифрових інтегральних схем у масштабі реального часу. При використанні системи обміну даними для управління

виробництвом, технологічним процесом чи транспортом ефективність систем може бути підвищена. В загальному випадку у зв’язку з розвитком архітектури міжпроцесорних систем основною задачею є побудова БС різноманітних видів.

Електронно-обчислювальна система зазвичай будується за допомогою підключення швидкодіючої пам'яті до процесора, що є, як правило єдиним, і обираючи довільні команди, виконує декодування. Окрім цього, система містить пристрої введення і виведення інформації. Над даними, що поступають в систему, виконуються однакові команди кожним процесором багато разів. Розмір матриці процесорів вказує на обсяг даних, що потрібно опрацювати.

Матриці мають деякі обмеження, що пов’язані перш за все з їх розмірами. Саме тому необхідно мати на увазі й інші види мереж, такі як сочевицеподібні, деревовидні, х–деревовидні, пірамідальні структури. Для випадка БС граф є відображенням набору підсистем, що являються одним мультипроцесором, окремими процесорами тут є вузли. Перевагами великих матричних систем є простота виконання, адже кожний процесор виконує одну команду. Для досягнення максимальної продуктивності в даних системах необхідно розглядати задачі, в яких є довгі ланцюжки однотипних операцій, тобто існує паралелізм даних.

Для досить великого числа задач, що необхідно виконати,дерево є достатньо ефективним засобом. В даних задачах виконується сортування інформації, її порівняння, а також реорганізація. Також хорошу структуру мають матриці, що використовуються для локальної передачі інформації[9].

Багатопроцесорні системи забезпечують величезний потенційний ріст продуктивності і обчислювальної потужності. Дійсно, будь-який граф, вузлами якого є окремі процесори, а дугами – безпосередні зв'язки між ними, зараз можна розмістити в конкретній багатопроцесорній системі. Саме тому

програма, що розробляється в проекті, є актуальною не лише на даний момент, але й у подальшому.

2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Програмне забезпечення та бібліотеки.

Проаналізувавши програмне забезпечення, для дипломного проекту було обрано фреймворк JUNG з мовою програмування Java.

Java – не єдина об’єктно–орієнтована мова програмування, але її особливістю є те, що вона відразу розроблялася з урахуванням принципів ООП (на відмінно від C++ і Object Pascal, де об’єктна орієнтація є доповнення до мов C і Pascal відповідно). Ось чому в Java все реалізовано у вигляді об’єктів – потоки виконання і потоки даних, робота з мережею чи зображенням, обробка помилок і т. п[4].

В Java було замінено множинне успадкування альтернативним підходом, а саме спеціальним типом “інтерфейс”. Там реалізовано строгу типізацію, тобто тип кожної змінної і будь-якого виразу уже відомий на етапі компіляції. Такий підхід полегшує виявлення помилок, адже компілятор відразу повідомляє про помилки і вказує місце в коді. Для підвищення надійності коду виконується пошук виняткових ситуацій під час виконання програми.

Ще однією перевагою мови програмування Java є можливість вивільнення пам’яті. У процесі розробки програміст створює об’єкти, працює з ними, а потім виникає необхідність видалити їх, щоб очистити пам’ять. В C/C++ це потрібно робити самостійно безпосередньо з програми. При такому підході є декілька недоліків: або видалити об’єкт, який ще необхідний, або ж пропустити непотрібний. Віртуальна машина мови Java самостійно підраховує кількість посилань на кожний об’єкт, і якщо вона дорівнює нулю, то такий об’єкт обробляється збирачем сміття.

Java – одна з найпопулярніших мов програмування в світі протягом останніх років. На відміну від декількох інших мов програмування, вплив Java не тільки не зменшився з часом, а, навпаки, збільшився. З моменту

першого випуску вона знаходиться на одному з перших місць програмування додатків для Інтернету. І кожна наступна версія лише зміцнювала цю позицію. Велика частина сучасного коду написана на Java. І це свідчить про особливе значення мови Java для програмування. Одна з основних причин успіху Java – її гнучкість. Починаючи з першої версії, ця мова безперервно адаптується до змін в середовищі програмування та підходам до написання програм. А найголовніше – вона не просто слідує тенденціям в програмуванні, а допомагає їх створювати. Здатність Java адаптуватися до швидких змін в обчислювальній техніці є причиною, по якій ця мова програмування тривалий час залишається успішною[10].

2.2. Мова програмування Java

Офіційною датою створення мови Java вважається 23 травня 1995 року, після випуску компанією Sun першої реалізації Java 1.0.

Мова програмування Java в багатьох аспектах схожа до мови C ++, яка тісно пов’язана з мовою C. Саме тому Java успадкувала чимало від обох цих мов. Від C було перейнято синтаксис, а від C ++ – об’єктно–орієнтовані властивості. Також у мові Java є чітка стандартизація, що прописана в офіційній документації[9].

Java має безліч стандартних бібліотек, що прискорюють розробку і значно підвищують надійність коду[10]:

1) "збирач сміття" – надійний захист від витоку оперативної пам'яті;

2) наявність великої кількості IDE дозволяють вести грамотну і зручну розробку;

3) зворотня сумісність – тобто додатки, написані на більш ранніх версіях Java, з великою ймовірністю працюють на сучасних версіях компілятора;

4) мультиплатформність – додаток запускається на будь-яких операційних системах, Windows, Linux, Mac OS і з невеликими

доповненнями на Android OS, тобто вибравши даний шлях реалізації програмного коду і після внесення ряд невеликих змін, ми отримуємо повноцінний мобільний додаток під Android OS, де допрацювавши графічну частину, можна отримати додатки під будь-які мобільні ОС з мінімальними затратами, і при певній необхідності мова Java дозволяє створити повноцінний WEB додаток, для обслуговування бази через інтернет. Java дає дуже широкий і гнучкий функціонал, зокрема дозволяє економити час розробника;

5) чітка ієрархічна структура ООП;

6) цікавою особливістю JVM є можливість писати будь-якими мовами програмування, які створюють коректний байт код. Тобто якщо якийсь модуль цього додатка необхідно буде реалізувати іншою мовою програмування, для кінцевого користувача це не викличе додаткових проблем;

7) Java має відкритий код під ліцензією GPL;

8) простою зміною драйвера можна змінити використовувану базу даних (БД) без серйозних змін коду;

9) JVM дуже добре адаптована під багатопотоковість і активно її використовує, що компенсує втрату ресурсів.

Користувач XI століття звик до зручного інтерфейса, для роботи в консолі потрібно мати додаткові навички. Саме тому важко уявити сучасну програму без Graphical user interface (GUI). Перевагами GUI є зручність введення вхідних даних за допомогою візуальних компонентів, таких як кнопки, текстові поля, списки і т.д[10].

У зв’язку з розвитком мов програмування виникло безліч бібліотек для роботи з набором візуальних компонент. Бібліотека в програмуванні – це набір класів та інтерфейсів, призначених для вирішення певних завдань.

2.3. Порівняння біблітек AWT та Swing

Початковим варіантом графічного інтерфейсу Java була бібліотека Abstract Window Toolkit (AWT), методи якої викликались з бібліотек, написаних на мові C. Методи бібліотеки AWT оперують графічними компонентами операційного середовища. Таким чином, програми, написані на Java, схожі на програми, що написані іншими мовами програмування, але водночас при запуску на інших платформах є ймовірність виникнення розбіжностей шрифтів чи компонент.

Для забезпечення мультиплатформенності AWT було прийнято рішення уніфікувати інтерфейси викликів компонент. Внаслідок цього їх функціональні можливості зменшились і кількість компонент стала меншою. До недоліків можна віднести відсутність в AWT таблиць, також відсутня підтримка відображення іконок. Але, незважаючи на все вище перераховане, пакет java.awt можна використовувати у необхідних цілях[11].

Елементи бібліотеки AWT самі по собі не виконують роботи. Запити, що надходять до компонент AWT, перенаправляються до операційної системи, тобто саме ОС виконує роботу.

На сьогоднішній день за допомогою бібіліотеки AWT написати візуально красивий та зручний інтерфейс буде важко. На її продуктивності негативно відображається те, що використані ресурси звільняються автоматично, що також ускладнює архітектуру.

Згодом компанією Sun було розроблено бібліотеку Swing, методи і класи якої написані на Java. На відмінно від AWT, бібліотека Swing підтримує різноманітні стилі. Функціональність набора стандартних компонентів значно ширша, ніж в AWT. Було добавлено легковагові (lightweight) компоненти, але найважливішою відмінністю і те, що ОС не пов’язана з компонентами Swing. Саме тому підвищились стабільність і швидкість[10].

Зважаючи на вище перераховані переваги, було прийнято використовувати бібліотеку Swing.

2.4. Бібліотека Swing

Swing є найпоширенішою бібліотекою для створення графічного інтерфейсу й завдяки реалізації відомого шаблону проектування Model–View–Controller (MVC). MVC зберігає код, що відповідає за зовнішній вид, окремо від кода, що оброблює дані, і окремо від кода, що описує реакцію на взаємодію і виконує зміни.

В основі шаблона проектування MVC закладено, що нехай кожний аспект UI має справу лише з тим, що він добре знає. Тобто головним принципом є те, що візуальні компоненти відображають дані, а інші класи управляють ними.

MVC складається з трьох частин[4]:

1) Модель (model) зберігає дані компонента і дозволяє легко, не звертаючись до самого компонента, змінювати чи отримувати ці дані. Наприклад, відкриваючий список дозволяє вивести на екран перелік елементів. Замість того, щоб включати методи для маніпуляції елементами списка в клас розкриваючого списка, можна представити окремий клас, що працює виключно з даними. Такий підхід дозволяє розробнику зосередитися саме на тій задачі, якою він займається в даний момент: можна спочатку підготувати дані, а потім уже передати їх списку для виведення на екран. Зберігання даних окремо від самого компонента також дозволяє змінити структуру даних моделі, не змінюючи функції компонента.

2) Вигляд (view) виводить дані на екран для представлення їх користувачу. Відділення вида від даних дозволяє представляти одні і ті ж дані абсолютно різними способами. Наприклад, текст формата HTML(Hypertext Markup Language – гіпертекстова мова розмітки) можна вивести в різному вигляді: провести розмітку документа, розмістити зображення і посилання, використовувати різні шрифти, а можна показати HTML–документ як код, який складається з набора тегів і текста серед них.

3) Контролер (controller) визначає, як повинні реагувати вид і дані моделі у відповідь на дії користувача. Наявність в MVC контролера дозволяє використовувати одні й ті ж дані і види в різних цілях. HTML–сторінка, наприклад, може відображатися в браузері або у візуальному середовищі створення сторінки. Списку, що розкривається, не будуть зайвими декілька контролерів: один для списка, а інший для списка, що може редагуватися.

Для кращого пояснення взаємозв'язку складових MVC на рисунку 1.

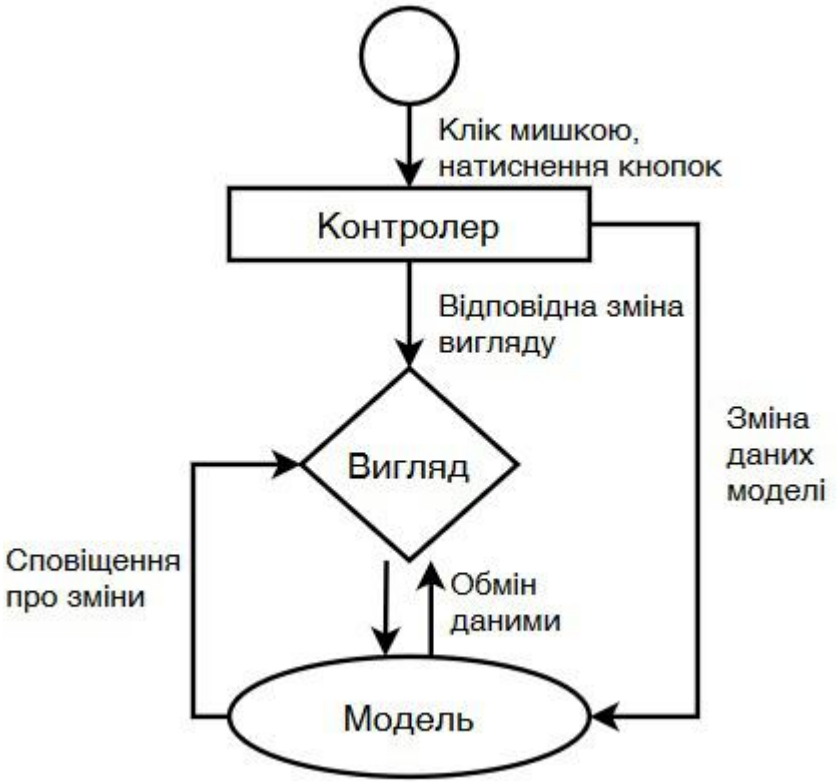


Рисунок 1 – Взаємодія між моделлю, виглядом і контролером

Кожен інтерфейс користувача розглядає три наступні аспекти[10]:

1) Елементи інтерфейсу – це основні візуальні елементи, які бачить користувач і з якими взаємодіє.

2) Макети – вони визначають, який вигляд матимуть елементи інтерфейсу користувача на екрані.

3) Поведінка – це події, які відбуваються, коли користувач взаємодіє з елементами інтерфейсу, наприклад при натисненні на клавішу.

Всі swing–програми повинні включати в себе контейнери першого рівня:

- JFrame;
- JApplet;
- JWindow;
- JDialog.

Ці контейнери знаходяться на вершині ієрархії контейнерів, що зображена на рисунку 2 і містять в собі інші, "легковагові" контейнери і елементи управління.

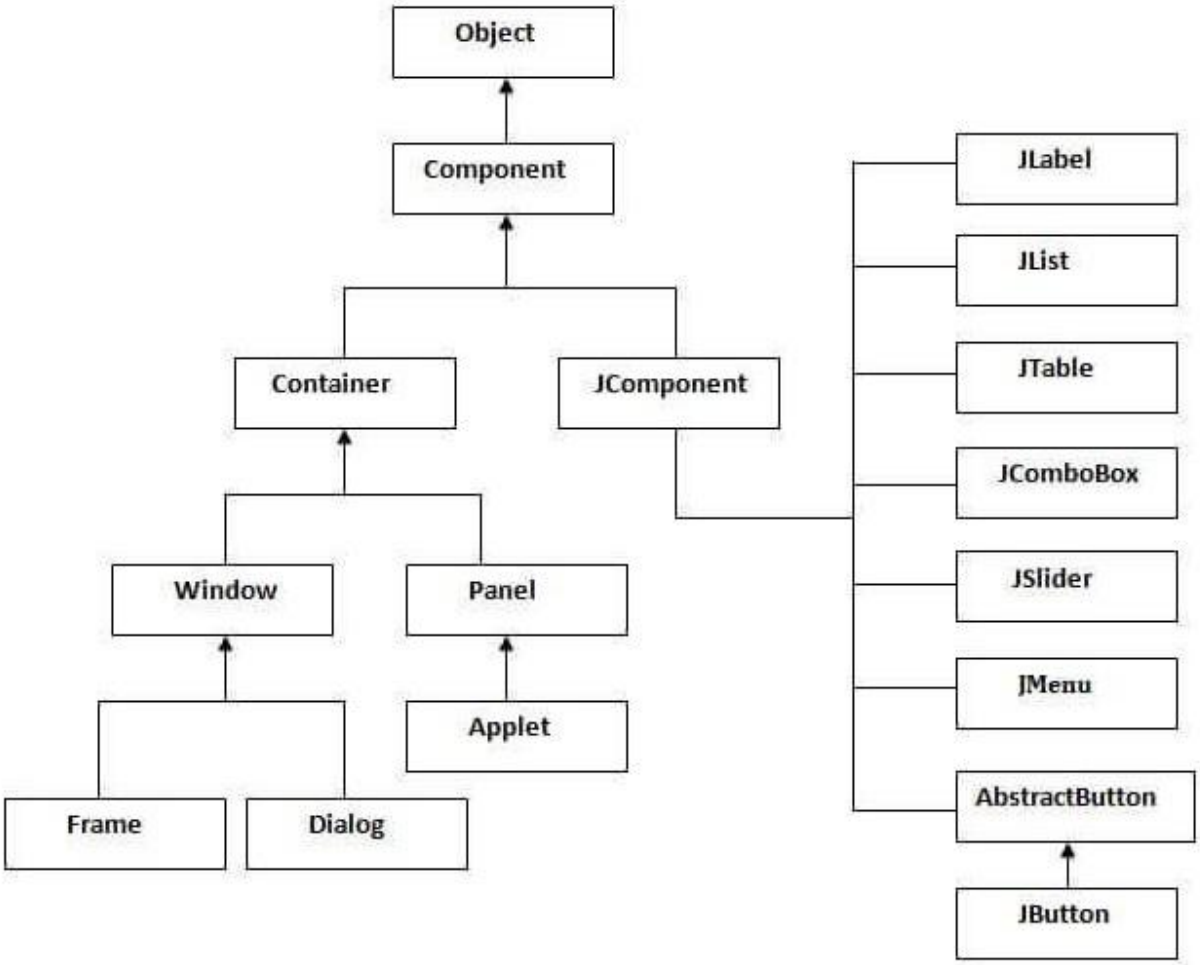


Рисунок 2 – Ієрархія класів бібліотеки Swing

2.4.1. Компоненти і контейнери.

Існує два типи елементів графічного інтерфейсу[11]:

1) JComponent (компонент) – це елементарні суб'єкти GUI, такі як Button, Label і TextField. JComponent містить метод add (), який дозволяє додавати інші компоненти.

2) Container (контейнер), такі як Frame і Panel, використовуються для зберігання компонентів у певному макеті (наприклад, FlowLayout або GridLayout). Контейнер також може мати інші контейнери.

На рисунку 3 є три контейнери: Frame і дві Panels. Frame є контейнером верхнього рівня даної програми. Frame має рядок заголовка, що містить іконку, заголовок і кнопки мінімізації/ розгортання/ закриття, а також додаткову панель меню і область відображення вмісту. Panel – це прямокутна область, яка використовується для групування відповідних компонентів графічного інтерфейсу в певному форматі. Кадр верхнього рівня містить дві Panels. Там є п'ять компонентів: Label (надає опис), TextField (для введення тексту), і три Buttons (для того, щоб користувач запускав певні запрограмовані дії).

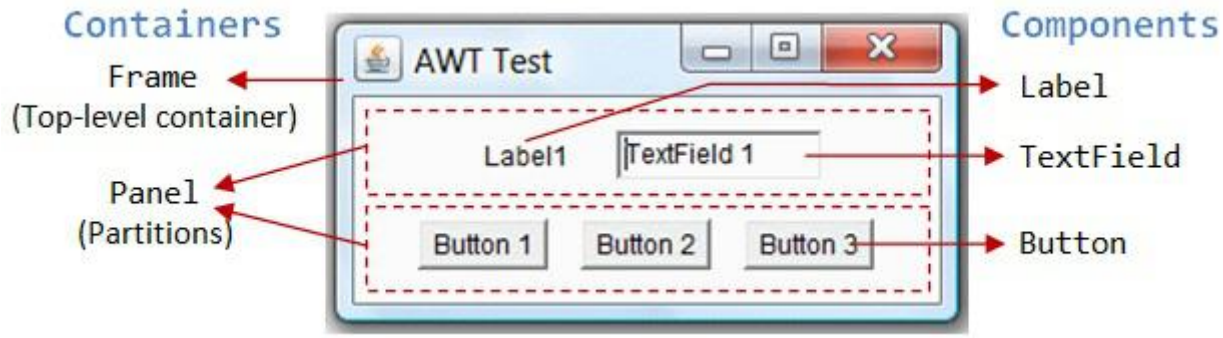


Рисунок 3 – Приклад використання контейнерів

В GUI компонент повинен зберігатися в контейнері. Потрібно ідентифікувати контейнер для утримання компонентів. Кожен контейнер має метод, який називається add (Component <назва>). Контейнер може викликати <назва>.add (aComponent), щоб додати aComponent.

2.4.2. Основні компоненти

Одним з основних компонентів бібліотеки Swing є JLabel, яка встановлюється в потрібному місці, покращує вигляд інтерфейсу і описує інші компоненти. Вона використовується не лише для відображення тексту, а й майже завжди для зображення. До основних методів відноситься установка текста і зображення, вирівнювання компонентів, які описує мітка:

1) get/setText () – отримати/установити текст в мітці;
2) get/setIcon () – отримати/установити зображення в мітці;
3) get/setHorizontalAlignment () – отримати/установити горизонтальну позицію текста;
4) get/setVerticalAlignment () – отримати/установити вертикальну позицію текста;
5) get/setDisplayedMnemonic () – отримати/установити мнемоніку для мітки[10].

Основним активним компонентом в Swing є JButton. Для кожної кнопки визначається послідовність дій, яка виконається у разі натиснення на неї. Методи, що використовуються для JButton, аналогічні методам JLabel:

Для JButton також існують методи станів. Стан – це властивість, що описує компонент, зазвичай має значення true/false. Інколи використовуються декілька методів станів. Для відображення зміни станів кнопки використовуються різні зображення.

Основним текстовим компонентом в Swing є JTextField, що дозволяє користувачу вводити текст в UI. В полі можна вводити, видаляти та виділяти текст, переміщати курсор. За допомогою метода get/setText () розробник отримує/установлює текст всередині JTextField.

Розглянуті вище мітка, кнопка і текстове поле зазвичай розміщуються в JFrame. Клас JFrame – це контейнер, куди можна додавати інші компоненти для їх організації і представлення користувачу. Цей компонент бібліотеки і

має властивості вікна операційної системи: мінімалізація/максималізація, зміна розмірів і переміщення. Ось деякі з методів для зміни властивостей JFrame[4]:

1) get/setTitle () – отримати/установити заголовок фрейма;

2) get/setState () – отримати/установити заголовок фрейма(мінімізація, максималізація);

3) is/setVisible () – отримати/установити видимість фрейма, тобто відображення на екрані;

4) get/setLocation () – отримати/установити місцезнаходження у вікні, де фрейм повинен з’явитися;

5) get/setSize () – отримати/установити розмір фрейма;

6) add () – додати компоненти до фрейма.

2.4.2. Обробка подій

Зміна стану об'єкта називається подією, тобто подія описує зміну стану джерела. Події генеруються в результаті взаємодії користувача з графічними компонентами інтерфейсу користувача.

Події можна розділити на дві категорії[10]:

1) Події переднього плану – це події, що вимагають безпосередньої взаємодії користувача. Вони генеруються як наслідок взаємодії людини з графічними компонентами в графічному інтерфейсі користувача. Наприклад, натискання кнопки, переміщення миші, введення символу через клавіатуру, вибір елемента зі списку, прокручування сторінки тощо.

2) Фонові події – це події, вимагають взаємодії кінцевого користувача. Наприклад, переривання операційної системи, збій апаратного або програмного забезпечення, закінчення таймера та завершення операції.

Обробка подій – це механізм, який вирішує, що має відбутися, якщо настає подія. Java використовує модель делегування подій, що визначає стандартний механізм генерування та обробки подій.

Модель делегування подій має наступних ключових учасників.

1) Source – це об'єкт, на якому відбувається подія. Джерело відповідає за надання інформації про подію, що сталася, її обробнику.

2) Listener – обробник подій, який відповідає за генерування відповіді на подію. З точки зору реалізації Java, слухач також є об'єктом. Слухач чекає, поки він отримає подію. Після отримання події слухач обробляє подію, а потім повертається.

Перевага цього підходу полягає в тому, що логіка інтерфейсу користувача повністю відокремлена від логіки, що генерує подію. Елемент інтерфейсу користувача може делегувати обробку події окремим фрагментом коду.

Як вказано в документації на API, методи для зміни розмірів і форми фреймів слід шукати в класі Component (який є предком всіх об'єктів графічного інтерфейсу) і в класі Window (суперкласі класу Frame). Наприклад, метод show (), який використовується для відображення фрейму на екрані, знаходиться в класі Window, а в класі Component є метод setLocation (), що дозволяє змінити місце розташування компонента. Координати фрейму, що задаються методами setLocation () і setBounds (), обчислюються відносно екрана. Координати інших компонентів всередині контейнера визначаються відносно самого контейнера[4,10].

2.5. Бібліотека JUNG

JUNG – Java Universal Network / Graph Framework – це бібліотека програмного забезпечення, яка надає мову для моделювання, аналізу та візуалізації даних, які можуть бути представлені у вигляді графіка або мережі. Вона написана на Java, що дозволяє додаткам на основі JUNG використовувати велику кількість вбудованих можливостей Java API, а також інших існуючих сторонніх бібліотек Java[5].

Архітектура JUNG розроблена для підтримки різних уявлень об'єктів і їх відносин, таких як орієнтовані і неорієнтовані графи, графи з паралельними ребрами і гіперграфи. Він надає механізм для анотування графіків, об'єктів і відносин з метаданими. Це полегшує створення аналітичних інструментів для складних наборів даних, які можуть досліджувати відносини між об'єктами, а також метадані, пов'язані з кожним об'єктом і відношення.

JUNG також надає середовище візуалізації, яке дозволяє легко створювати інструменти для інтерактивного дослідження багатопроцесорних систем. Крім того, передбачені механізми фільтрації, які дозволяють користувачам зосередити свою увагу або свої алгоритми на певних частинах графа[10].

Перевагою використання бібліотеки JUNG це те, що єдиним обмеженням на розміри побудованого графа, в тому числі і на кількість вершин чи ребер, є параметри комп’ютера, на якому запускається віртуальна машина Java.

JUNG реалізує типи графів за допомогою інтерфейсів Java (що визначають, які методи повинні забезпечувати певну реалізацію інтерфейсу), абстрактних класів (що надають узагальнену реалізацію інтерфейсів для прискорення розробки, але не можуть бути створені користувачами), і класів (що користувачі створюють і використовують).

Пакет graph містить специфікації (у вигляді Java – інтерфейсів), на різних рівнях абстракції, для графів, вершин і ребер[12].

1) Інтерфейси.

Інтерфейси ArchetypeGraph, ArchetypeVertexi, ArchetypeEdge задають поведінку узагальнених графів, вершин і ребер; вони призначені для охоплення всіх типів графіків, включаючи орієнтовані і неорієнтовані графи, гіперграфи і графи з паралельними ребрами. Всі реалізації графа, вершини і ребра повинні реалізовувати відповідні методи (або успадковуватися від цих

інтерфейсів). Методи, перераховані вище, є доступними для об'єктів, які реалізують ці методи передачі даних.

Інтерфейси Graph, Vertex та Edge успадковуються від Archetype і визначають поведінку для графів, в яких кожне ребро з'єднує рівно дві вершини; це дозволяє визначити ряд додаткових методів.

Інтерфейси Directed визначають поведінку і можливості орієнтованих графів і ребер. A DirectedEdge– це тип ребра, який впорядковує вершини. DirectedGraph – це інтерфейс для реалізації графа, який має безліч ребер типу DirectedEdge.

Інтерфейси UndirectedGraph і UndirectedEdge являються відповідними інтерфейсами для неорієнтованих графів і ребер.

2) Абстрактні класи.

Класи AbstractSparseGraph, AbstractSparseVertex і AbstractSparseEdge призначені для розріджених графів (у яких кількість ребер максимально близька до кількості вершин). Для реалізації графа з обширними зв’язками ці класи не є найкращим вибором.

3) Класи впровадження

Класи DirectedSparse {Graph, Edge, Vertex} і UndirectedSparse {Graph, Edge, Vertex} розширюють абстрактні класи для строго орієнтованих і неорієнтованих графів.

JUNG надає механізми для викладення та надання графіків. Поточні реалізації рендерера використовують Java Swing API для відображення графіків, але вони можуть бути реалізовані з використанням інших інструментальних засобів.

Загалом, візуалізація виконується за допомогою:

Макет, який приймає графік і визначає місце розташування, на якому буде намальована кожна з його вершин.

Компонент (Swing), в який передаються дані. (Поточні реалізації використовують VisualizationViewer, який є розширенням класу Swing JPanel.)

Рендерер, який бере дані, надані макетом, і малює вершини і ребра у наданий компонент.

Таким чином, вибираючи один з цих трьох, можна координувати креслення. Реалізація за замовчуванням проходить макет, запитуючи її про розташування вершин, а потім малює їх окремо з рендерером всередині компонента Swing. Крім того, інфраструктура GraphDraw спрощує багато з цих перетворень, упакуючи візуалізацію Viewer, Renderer і Layout разом. Після цього користувачі можуть налаштувати цей переглядач відповідно. (Зразок коду доступний у документації GraphDraw.)

JUNG також включає в себе утиліти та класи підтримки, які полегшують налаштування візуалізації графіка. Наприклад, FadingVertexLayout надає механізм, який можна використовувати для створення ефектів затухання, коли вершини відфільтровуються і потім відновлюються[5].

2.6. Елементи теорії графів.

За останні роки у зв'язку з розвитком теорії графів на перший план стають проблеми аналізу багатопроцесорних обчислювальних систем. Нехай V – непуста множина, наприклад {v1, v2, v3, v4, v5}. Для даного випадку множиною всіх його двохелементних підмножин V(2)[13]:

$$V^{(2)} = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_1, v_5\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_5\}\}.$$

Візьмемо довільно деяке $E \subseteq V^{(2)}$:

$$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_4, v_5\}\}.$$

Пару $\langle V, E \rangle$ називають неорієнтованим графом G, в якому V – це множина вершин, а E – множина ребер, що є підмножиною множини V(2). В більш компактній формі це визначення зазвичай звучить так: пара $\langle V, E \rangle$ називається неорієнтованим графом, якщо V – непуста множина елементів, що називається вершинами, а E – множина невпорядкованих пар різних елементів із , що називаються ребрами.

Для запису різноманітних відношень в теорії графів користуються позначеннями VG або V(G) для множини вершин і EG або E(G) для множини ребер графа G.

Наочним способом представлення графа є рисунок (діаграма), на якому вершини зображаються точками, кругами чи іншими фігурами, а ребра – лініями, що з'єднують зображення вершин реберної пари. Форма і розміри зображення значення не мають. Важливо тільки, щоб воно відповідало множинам V і E. На рисунку 4 дано варіанти такого представлення вище описаного графа.

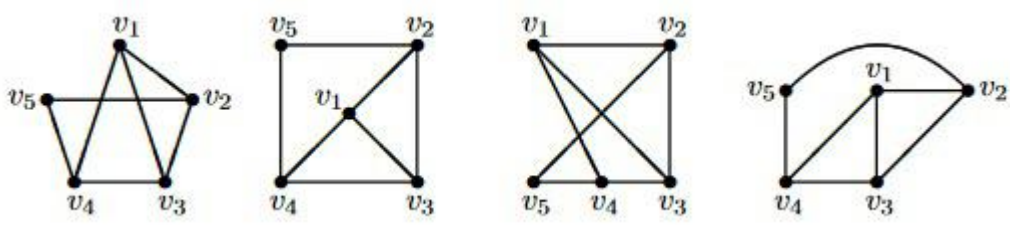


Рисунок 4 – Приклад зображення графа

Основні характеристика графа і його елементів

Дві вершини, що утворюють $\{v_i, v_j\}$, називають його кінцями , а про ребро кажуть, що воно з'єднує v_i і v_j. Вершини v_i і v_j є суміжними. Суміжними називають і ребра зі спільною вершиною. Ребро і вершину, що є його кінцем, називають інциндентними. Наприклад , в графі на рисунку 4 вершини v_1 і v_2 є суміжними, а вершини v_1 і v_3 не суміжні.Ребра $\{v_1,v_2\}$ і $\{v_1,v_3\}$ – суміжні, а ребра $\{v_4,v_5\}$ і $\{v_2,v_3\}$ не суміжні. Вершина v_3 і ребро $\{v_2,v_3\}$ інцидентні[13].

Число ребер, що інцидентні вершині v, визначають степінь вершини, яка позначається deg v. Таким чином, в графі на рисинку 4 deg v_1 = deg v_2 = deg v_3 = deg v_4 = 3, а deg v_5 = 2. Вершину v називають ізольованою, якщо deg v = 0, і кінцевою, якщо deg v = 1. Ребро, інцидентне кінцевій вершині, називається кінцевою. Список степенів всіх вершин називають степенневою послідовністю графа.

Множина вершин, що суміжні з вершиною v, позначають adj v. Наприклад, в графі на рисунку 4 adj v_1 = {v_2,v_3,v_4}.

Важливими кількісними характеристиками графа є: число вершин n=|V|, що визначає порядок графа і число ребер m=|E|. Граф з n вершинами і m ребрами називається (n, m)–графом.

Історорично першою теоремою теорії графів є твердження Ейлера, що пов'язує кількість ребер, вершин і їх степенів.

Теорема 1. Сума степенів вершин (n, m)–графа рівна подвоєному числу його ребер: $\sum_{i=1}^{n} \deg v_i = 2|E| = 2m.$

Доведення звучить так: оскільки будь-яке ребро інцидентне двом вершинам, то в сумі степенів всіх вершин графа кожне ребро враховується двічі.

Як наслідок, в будь-якому графі число вершин непарної степені парне. Нехай і множини вершин парної і непарної степені відповіно. Очевидним є те, що $\sum_{v \in V_1} \deg v + \sum_{v \in V_2} \deg v = 2m.$

Маємо два складових, сума яких парне число. Перший доданок парний, тому друге число має бути парне, а це при додаванні непарних чисел можливе лише тоді, якщо їх кількість парна[14].

Ізоморфізм графів

На рисунку 4, де зображені чотири різних зображення одного й того ж графа, показано різноманітність образів при графічній інтерпретації графа.

Для того, щоб визначити чи є різні зображення графа відображенням для одного графа, використовується ізоморфізм[14].

Ізоморфізмом називають взаємно однозначну відповідність між множинами вершин двох графів G_1 і G_2 , що зберігають відношення суміжності, а самі графи називають ізоморфними. Відображаючи це, пишуть: $G_1 \cong G_2$ або $G_1 = G_2$.

Ізоморфність графів на рис. 4 установити легко. Достатньо скласти списки вершин всіх графів, указуючи для кожної із них всіх її сусідок (множина adj v). Порівнюючи списки, можна визначити ізоморність графа. В даному випадку графи дійсно ізоморфні, до того ж їх еквівалентні вершини позначені однаково. Задача ускладнюється, якщо еквівалентні вершини графів мають різні номера. В якості приклада можна привести два ізоморфних графа G_1 і G_2, що зображені на рисунку 5. Для них описану вище процедуру роботи

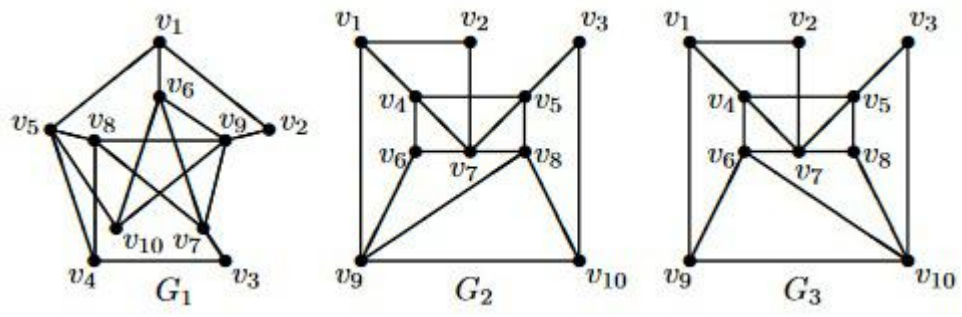


Рисунок 5 – Ізоморфні графи

зі списками прийдеться повторювати неодноразово, кожний раз змінюючи нумерацію вершин одного з графів, пока не буде виявлено ізоморфізм, тобто отримані ідентичні списки вершин. Доказати неізоморфізм графів G_1 і G_3 можливо, якщо виконати всі перевірки. Істотно зменшити їх кількість можна, якщо використовувати інваріанти.

Інваріантом називають деяку характеристику графа G, яка приймає одне і теж значення для будь-якого графа, ізоморфного G[14].

Інваріантами є: число вершин і число ребер графа, число вершин парної і непарної степені, степеннева послідовність та інші числові характеристики. В якості інваріантів можуть виступати властивості і особливості графа, такі як зв'язність, дводольність, наявність чи відсутність циклів і т.п[13].

Кількість графів

Визначимо число графів порядка n. На множині із n вершин можливо утворити C_n^2 різних невпорядкованих пар, що відповідають всім можливим ребрам графа. Тому будь-якому n-вершинному графу можна поставити у відповідність C_n^2 –розрядний двійковий код, кожний розряд якого відповідає певному ребру: якщо розряд дорівнює 1, то граф містить це ребро, якщо ж 0 – то не містить. Звідси слідує, що кількість графів порядка можна визначити як $l_n = 2^{C_n^2}$. Однак, якщо не враховувати розмітку вершин, яка принципово значення не має, а лише дозволяє описати зв'язки між вершинами, не всі ці графи є різними. Наприклад, існують $l_3 = 2^{C_3^2} = 8$ трьохвершинних графів, оскільки $G_2 \cong G_3 \cong G_4$ і $G_5 \cong G_6 \cong G_7$ і, якщо незважати на мітки вершин, різниця між графами в цих трійках зникне. Всі вони представлені на рисунку 6. Через ці фактори з'явилося таке поняття як абстрактний граф[15].

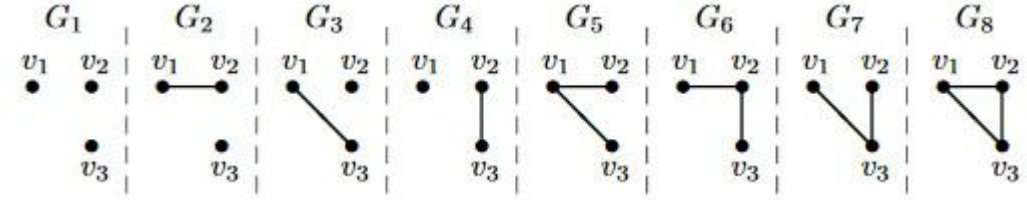


Рисунок 6 – Приклади тьрохвершинних графів

Зв'язні і незв'язні графи

Існує достатньо проста формула Пойа, що дає асимптотичну оцінку числа непомічених графів: $g_n \sim 2_n^2 / n!$. Згідно з формулою кількість непомічених графів приблизно в n! раз менша за кількість помічених[13].

Граф незв’язний, якщо множина його вершин розкладається на два або більше підмножин, що не перетинаються і не мають жодного ребра, кінці якого належать різним підмножинам. Ці графи називаються компонентами зв’язності. На рисунку 7 зображені незв’язані (G_1 – G_5) і зв’язані (G_6 – G_{11}) графи. Граф G_1 має чотири компоненти, G_2 – три компоненти, а G_3 ,G_4 і G_5 – дві, всі інші графи мають одну компоненту.

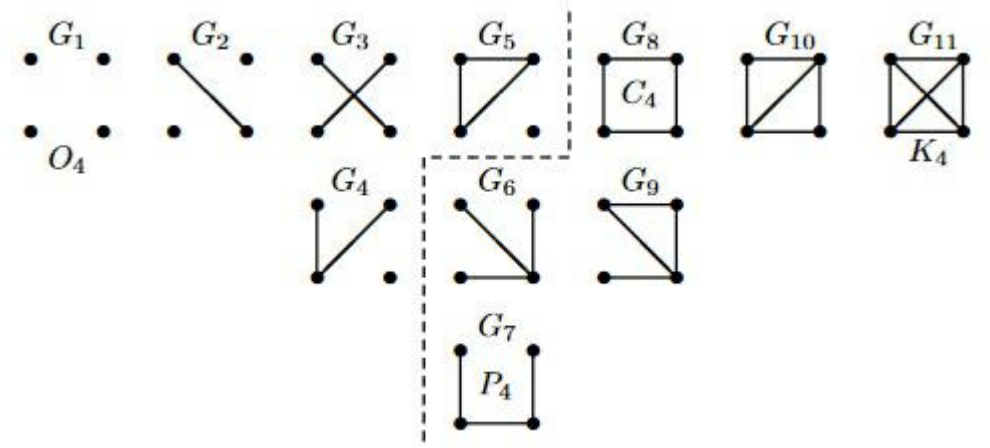


Рисунок 7 – Види графів

Серед незв’язаних графів, що зображені на рисунку вище, особливе положення займає граф G_1 , який взагалі не має ребер і називається пустим. Для таких графів часто використовується позначення O_n , де n – число вершин[14].

Серед зв’язних графів $G_6 - G_{11}$ також можна виділити граф G_{11}, що має максимальне для графа четвертого порядку число ребер. Він називається повним графом. Для таких графів використовується позначення K_n. Ясно, що число ребер в повному графі дорівнює кількості двохелементних підмножин множини V.

Для графів G_1, G_3, G_8, G_{11} характерним є те, щоб в кожному з них всі вершини мають однакові степені. Такі графи називають регулярними або однорідними. На рисунку 8 наведено приклади регулярних графів третьої, четвертої та п’ятої степені.

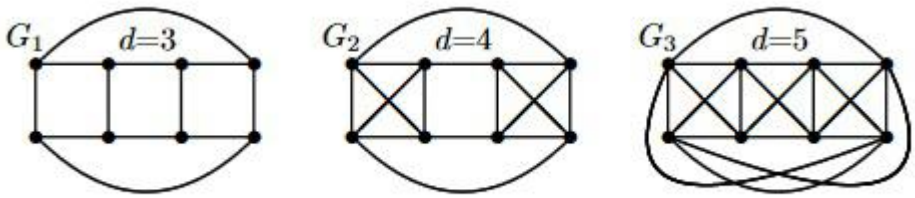


Рис. 1.8

Рисунок 8 – Приклад регулярних графів

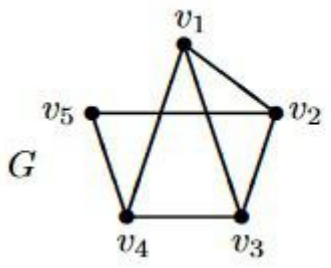
Ці графи третьої степені називають кубічними (G_{11} на рис. 7 та G_1 на рис. 8). Число ребер в регулярному графі степені d дорівнює $m = \frac{1}{2}nd$. Із цього слідує, що при непарному числі вершин регулярний граф може мати лише парну степінь. Тому кубічний граф має парну кількість вершин[14].

Матриця суміжності

Матриця суміжності - це симетрична квадратна матриця $A = \lfloor a_{i,j} \rfloor$ порядка n, в якій елемент $a_{i,j}$ дорівнює 1, якщо в графі є ребро $\{v_i, v_j\}$, тобто v_i і v_j суміжні, і 0, якщо такого ребра немає[16].

З цього слідує, що $\sum_{j=1}^{n} a_{i,j} = d(v_i)$ для будь-якого i, $\sum_{i=1}^{n} a_{i,j} = d(v_j)$ для будь-якого j та $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j} = 2m$, тобто кількість одиниць в будь-якому рядку дорівнює степені відповідної вершини графа, а загальна кількість одиниць дорівнює подвоєному числу його ребер.

В якості приклада на рис. 9 наведено граф G , його матриця суміжності A і степенева послідовність deg v_i .



$$\mathbf{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \deg v_i \\ 3 \\ 3 \\ 3 \\ 4 \\ 1 \end{array}$$

Рисунок 9 - Приклад графа з матрицею суміжності

Матриця суміжності пустого графа O_n складається з нулів, тобто $A(O_n) = 0_n$. Матриця суміжності повного графа K_n складається з одиниць, окрім діагональних елементів, що дорівнюють 0. Прийнято записувати це так: $A(K_n) = 1_n - I_n$. Якщо граф складається з незв'язний і має s компонент, то за допомогою перестановок рядок і стовпців, його матрицю можна привести до блочно-діагонального вигляду:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{ss} \end{bmatrix},$$

де кожний діагональний блок A_{ij} є матрицею суміжності компоненти s_i.

У випадку k-дольного грфа матриця суміжності може бути приведена тільки до блочного вигляду, коли на головній діагоналі стоять тільки нульові блоки:

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{A}_{12} & \dots & \mathbf{A}_{1k} \\ \mathbf{A}_{21} & \mathbf{0} & \dots & \mathbf{A}_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{k1} & \mathbf{A}_{k2} & \dots & \mathbf{0} \end{bmatrix}.$$

Блоки A_{ij}, де $i \neq j$, можуть розглядатися як приведені матриці суміжності дводольних підграфів k-дольного графа, що утворені вершинами і доль, при чому $A_{ij} = A_{ji}^T$. В якості приклада на рис. 10 дано трьохдольний граф з долями $\{v_1,v_2\}$, $\{v_3,v_4,v_5\}$,$\{v_6,v_7\}$ і його матриця суміжності A[17].

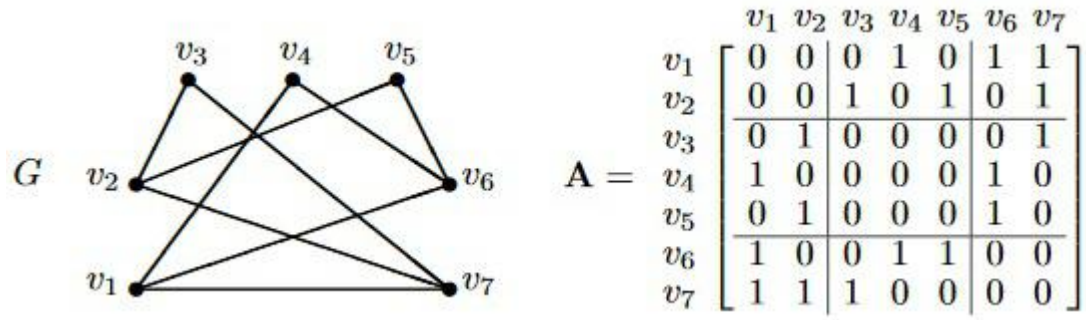


Рисунок 10 - Трьохдольний граф

Аналогом матриці суміжності є матриця Кірхгофа $K = \left[k_{i,j}\right]$, що визначена як квадратна матриця порядка n, елементи якої

$$k_{i,j} = \begin{cases} -1, & \textit{якщо вершини } v_i \textit{ і } v_j \textit{ суміжні}, \\ 0, & \textit{якщо вершини } v_i \textit{ і } v_j \textit{ не суміжні}, \\ \deg v_i, & \textit{якщо } i = j. \end{cases}$$

Зв'язок між матрицею суміжні А і матрицею Кірхгофа К має вигляд К=D-A, де $D = diag\left(\deg v_1, \deg v_2, ..., \deg v_n\right)$, тобто являється матрицею, діагональні елементи якої дорівнюють степеням відповідних вершин. Важлива особливість цієї матриці заключається в тому, що алгебраїчні доповнення всіх елементів матриці рівних між собою[15].

Оскільки ізоморфні графи відрізняються лише розміткою (нумерацією) вершин відповідного абстрактного графа, ясно, що матриці Кірхгофа можуть бути отримані одна з іншої шляхом деякої перестановки рядків і стовпчиків.

Матриця інцидентності

Матриця інцидентності - це прямокутна матриця B=[$b_{i,j}$] розмірності n×m, в якій елемент $b_{i,j}$ дорівнює 1, якщо вершина v_i інцидентна ребру e_j, і 0 в протилежному випадку. Рядки матриці B називають векторами інцидентності і позначаються наступним чином B_i. На рис. 11 зображений той же граф G, що і на рис. 12, і приведена його матриця інцидентності B[16].

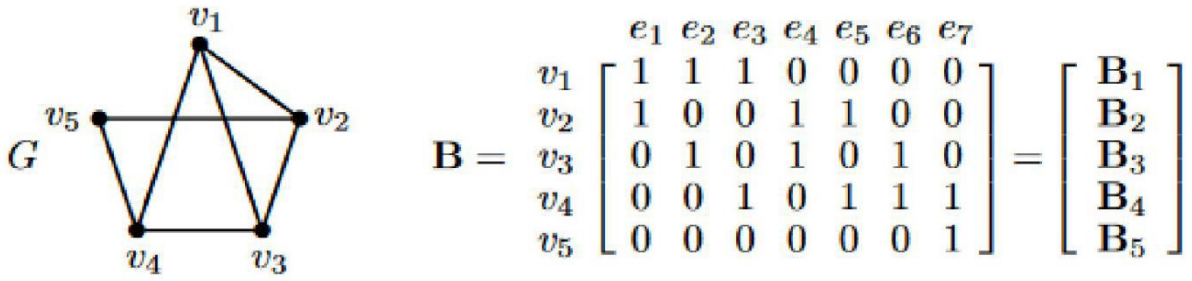


Рисунок 11 - Приклад графа з матрицею інцидентності

Циркулянтні графи реалізовані як комунікаційні мережі в паралельних обчислювальних системах ILLIAC–IV, MPP, Intel Paragon, Cray T3D, SONET. Вони використовуються також в якості структур багатомодульної високошвидкісної пам'яті обчислювальних систем. В даний час розширюються можливості практичного застосування циркулянтних мереж і їх узагальнень як основи структури в мультипроцесорних кластерних системах, в моделі small–world networks, оптичних мережах, моделях хімічних реакцій, клітинних нейронних мережах, що пояснюється високими показниками надійності, модульності і зв'язності цих графів. Важливою особливістю двовимірних і трьохвимірних циркулянтних графів є їх використання в теорії кодування при побудові досконалих кодів, що виправляють помилки[15].

Визначення 1. Нехай $s_1, s_2, ..., s_k, n$ – цілі числа, такі, що $1 \le s_1 < s_2 < ... < s_k < n$. Неорієнтований граф C з множиною вершин $V = \{0,1, ..., n-1\}$ і множиною ребер $E = \{(i, j) : |i - j| \equiv s_m \pmod n), m = 1, ..., k\}$, називається циркулянтною мережею[16].

Іншими словами, циркулянтним графом є графом Келі, в якого матриця суміжності - циркулянт. Елементи множини S = { s_1, s_2, ..., s_k} називаються твірними (хордами)[17]. Параметричний опис виду $(n; S)$ повністю визначає циркулянт n-ого порядка і розмірності k. Степінь циркулянта дорівнює $2k$, якщо $s_k = n / 2$. Якщо n парне і $s_k = n / 2$, то циркулянт має степінь $2k-1$. Кожне

$s_i \in S$, взаємно просте з n, призводить до гамільтонового цикла в графі. Приклади циркулянтів розмірностей 2 і 3 показані на рис. 12.

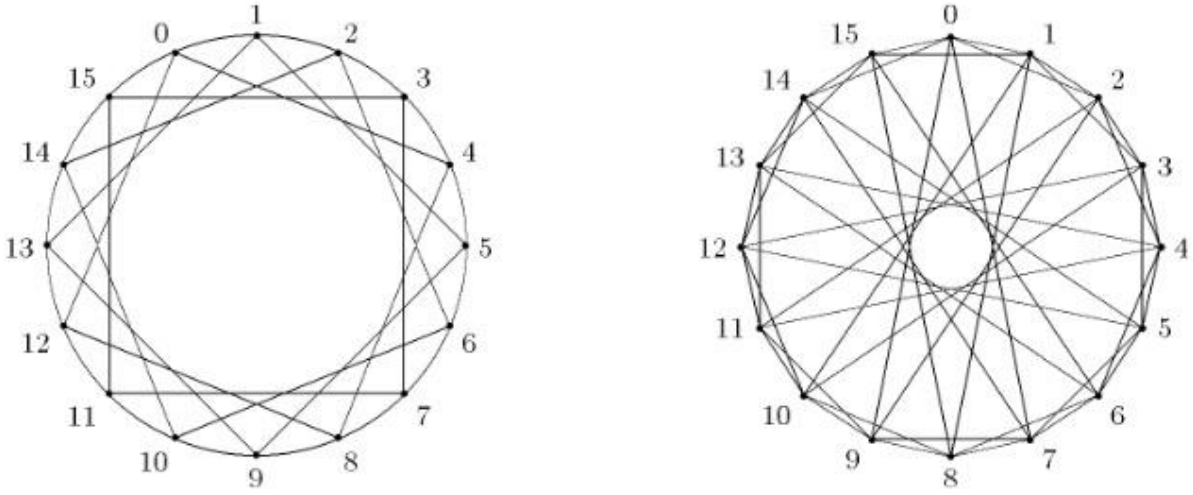


Рисунок 12 – Приклади циркулянтів

Граф називається зв’язним, якщо існує принаймні один шлях між будь-якими двома його вершинами. Надалі під (a, b) розуміється найбільший спільний дільник чисел a і b[18].

Лема 1. Якщо $(n, s_1) = 1$, то в циркулянтному графі $C(n; s_1, s_2)$ існує гамільтоновий цикл, який використовує тільки хорду s_1.

Тим не менш циркулянт може бути зв'язним і мати гамільтоновий цикл, але не мати гамільтового цикла, породженого однією з твірних (наприклад, граф $C(24; 3,4)$ При вирішенні проблем синтезу і вибору структур обчислювальних систем важливим є питання ізоморфізму графів. Із симетрії циркулянтів витікає наступна лема[19].

Лема 2. Циркулянти $C(n ; s_1,... s_i, ..., s_k)$ і $C(n ; s_1,... n - s_i, ..., s_k)$ ізоморфні.

Ця властивість дозволяє обмежитися розглядом циркулянтних графів з твірними, що не перевершує n / 2.

Лема 3. Якщо $(n,t)=1$, то $C(n;s_1,\dots s_i,\dots,s_k)$ і $C(n;t_{s_1},t_{s_2},\dots,t_{s_k})$ ізоморфні (твірні $ts_i, i=1,\dots,k,$ взяті по модулю n).

Важливими метричними характеристиками графа є діаметр і середня відстань, що відповідають максимальній і середній структурним затримкам в мережі.

Визначення 2. Діаметром графа С називається $d(n;S)=\max\limits_{i,\,j\in V} d(i,j),$ де $d(i,j)$ – довжина найкоротшого шляху між вершинами і та j, що належать С.

При фіксованому k нехай $D(n)$ означає точну нижню межу діаметра для будь-якого n. А також $d_{min}(n)=\min\limits_{S}\{d(n;S)\}\geq D(n)$ для будь-якого n.

Визначення 3. Середньою відстанню (або середнім діаметром) графа $C(n;S)$ називається $\bar{d}(n;S)=\sum\limits_{i,\,j} d(i,j)/(n(n-1))$.

Як показали дослідження, найкращими структурами обчислювальних систем по різним критеріям функціонування (структурної живучості, надійності та продуктивності) при однаковому числі обчислювальних модулів і ліній зв'язку у кожного модуля є структури з мінімальним діаметром і середньою відстанню.

Фундаментальна проблема теорії графів – синтез оптимальних графів – заключається в пошуку графів з мінімальним діаметром і/або мінімальною середньою відстанню серед регулярних графів із заданим степенем і числом вершин[20].

Розглянемо верхню межу максимально можливого числа вершин в циркулянтах. Для циркулянтного графа розмірності k нехай $P'(d,k)$ визначає число вершин, які знаходяться на відстані не більше ніж d від вершини 0, а $P(d,k)$ – верхню межу $P'(d,k)$. Нехай $S'(d,k)=P'(d,k)-P'(d-1,k)$ і $S(d,k)$ – верхня межа $S'(d,k)$. Значення $P(d,k)$ для циркулянта діаметра d і розмірності k отримано в результаті дослідження Ч. К. Вонгом і В. В.

Корнєєв. При виводі виразу $P(d,k)$ розглядається множина точок в k–вимірному Евклідовому просторі[21].

В циркулянті розмірності k функція $P(m,k)$ визначає максимальне число вершин, які можуть бути досягнені з будь-якої вершини графа за m кроків.

Визначення 4. Нехай d – додатнє ціле, $S=\{1,s_2,...,s_k\}$ – множина додатніх цілих, $m(d,S)=\max\{m:d(m,S)\le d\}$. Для будь-яких заданих цілих d і k буде справедливим вираз $M(d,k)=\max\{m(d,S):\exists S(|S|=k)\}$.

Теорема 1. Значення $M(d,k)$ є непарним числом при будь-яких $d\ge 1$ і $k\ge 1$.

Нехай n і k – додатні цілі, $d(n,k)$ – мінімально можливе натуральне d, таке, що існує множина твірних $S=\{1,s_2,...,s_k\}$, для якого $d(n;S)\le d$.

Визначення 5. Величини n і S циркулянтного графа $C(n;S)$ називаються оптимальними, якщо $d(n;S)=D(n)$, і субоптимальними, якщо $d(n;S)=D(n)+1$.

Визначення 6. Множина натуральних чисел θ називається оптимальною (субоптимальною), якщо кожне $n\in$ θ оптимальне (субоптимальне)[22].

Визначення 7. Циркулянтний граф $C(n;S)$ називається гранично оптимальним, якщо число вершин $C(n;S)$ на відстані m від вершини 0, де $m=0,1,...,D(n)-1$, дорівнює $S(m,k)$, а на відстані $D(n)$ – $(n-P(D(n)-1,k))$.

Таким чином, гранично оптимальний граф є найбільш щільним можливим графом для заданих n і k. Кожний гранично оптимальний граф є оптимальним, але не кожний оптимальний – граничним. Гранично оптимальні графи досягають точних нижніх меж діаметра і середньої відстані й відповідно мінімумів максимальної і середньої структурної затримки, максимумів зв'язності і надійності, мінімального числа кроків при реалізації

комунікаційних алгоритмів, але існують не для всіх величин n і k. Для важливого випадку k=2 вони існують для будь-якого числа вершин[22].

3. ПРОГРАМА ВІЗУАЛІЗАЦІЇ ДІАГНОСТИЧНИХ ГРАФІВ

Як було сказано вище, для візуалізації графа було обрано бібліотеку JUNG, для відображення інтерфейсу бібліотеку Swing і мову програмування Java.

3.1. Інструкція користувача

На рисунку 13 зображено вікно для вводу вхідних даних. Для реалізації візуально зручного вигляду цього вікна було використано візуальний дизайнер форм WindowsBuilder. Для того, щоб розмітити елементи GUI в необхідних місцях вікна, було використано Absolute Layout.

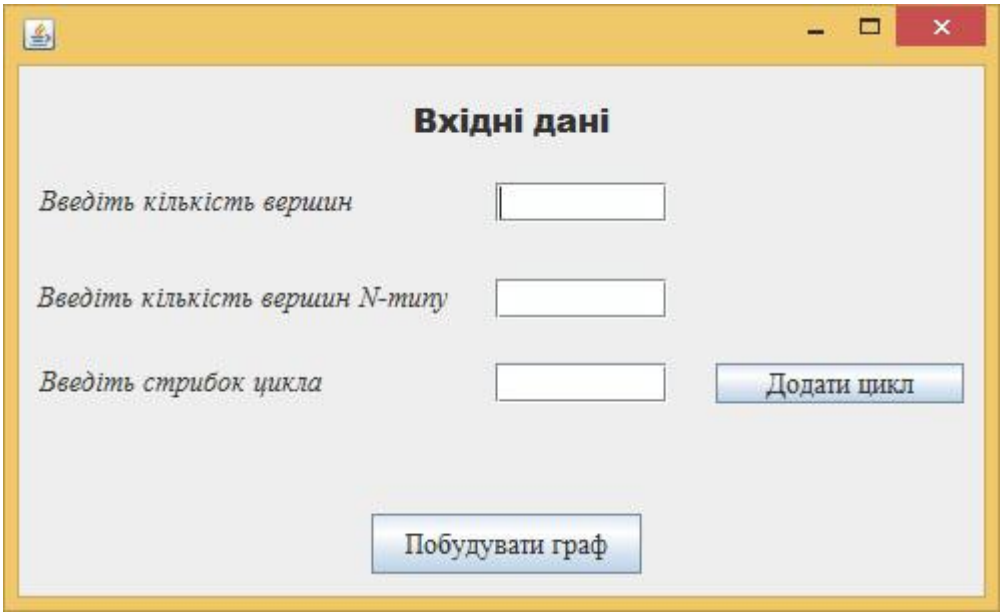


Рисунок 13 – Вікно вхідних даних

Вікно вхідних даних має такі компоненти:

– JLabel для оголошення назви вікна та опису дій, що очікуються від користувача. Було змінено шрифт за допомогою дизайнера форм;

– JTextField для полів вводу. Щоб ввести дані, необхідно клацнути курсором на одне з трьох полів. Для третього поля реалізовано можливість

введення більше ніж одного значення стрибка циклу, адже при побудові графів їх може бути декілька;

– JButton для двох кнопок: перша button1 для реалізації можливості додавання декількох стрибків циклу, а друга button2 для побудови графа.

Було створено два обробника подій: для натиснення на кнопку button1 – addCycles, а для button2 – actionListener. При натисненні button1 програма зчитує з відповідного поля вводу дані у масив arrayOfCycles. При натисненні на кнопку button2 дані з полів вводу зберігаються і відкривається вікно з побудованим графом.

Граф зберігається у поточну папку проекта у форматі png. На рисунку 14 зображено приклад діагностичного графа з 5 вершинами і двома циклами з стрибком 1 і 2.

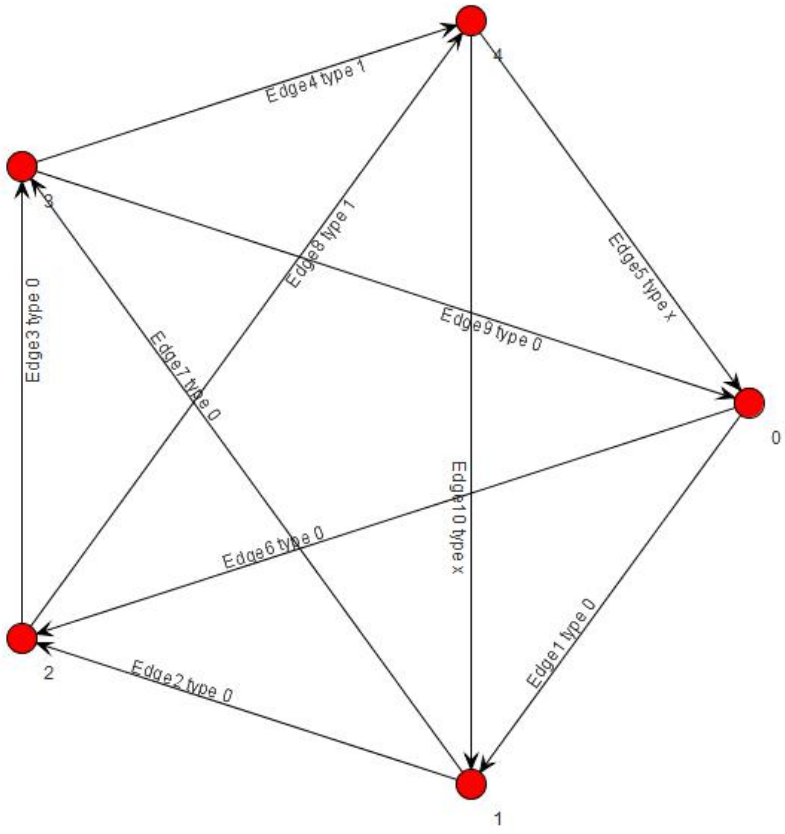


Рисунок 14 – Приклад діагностичного графа

На другому вікні було додано кнопку JButton button3. Клас ButtonEventListenerForRestart є обробником подій для цієї кнопки. При

натисненні button3 поля ввода даних обнуляються і з’являється вікно ввода вхідних даних.

Було реалізовано можливості масштабування і зміни місцезнаходження графа та його вершин окремо.

Для переміщення будь-якої вершини необхідно скористатися лівою кнопкою мишки і перетягнути її на потрібне місце вікна. Колір вершини зміниться і в консолі з’явиться повідомлення типу “Vertex 1 is now selected”. Приклад переміщення одиничної вершини показано на рисунку 15.

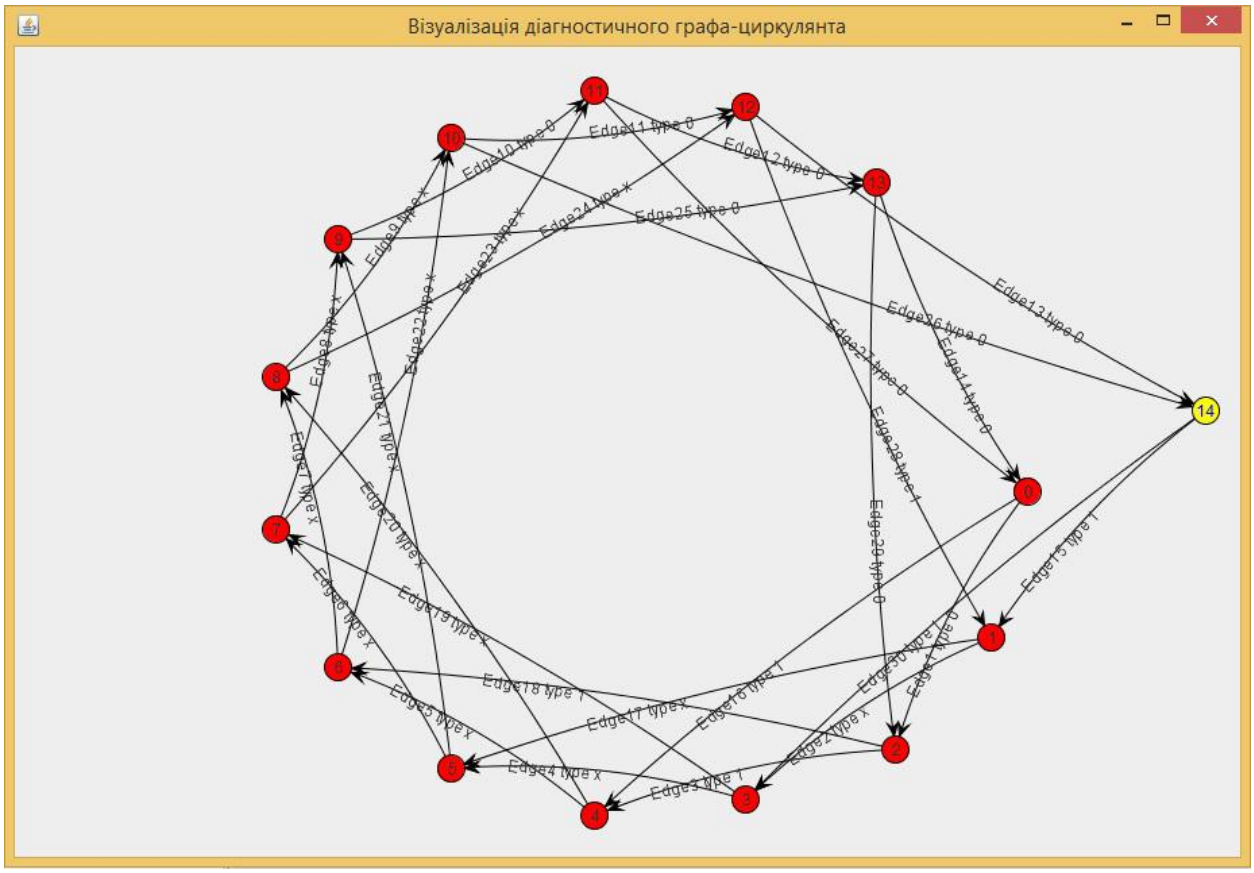


Рисунок 15 – Приклад переміщення вершини

Для невеликої кількості вершин і ребер можна побачити всі зв’язки, але для ілюстрації графа з достатньо великою кількістю ребер і вершин можна масштабувати зображення за допомогою колеса прокрутки (рис. 16).

Для того, щоб перемістити ввесь граф, незмінюючи розташування ребер і вершин між собою, необхідно виділити лівою клавішою мишки всі вершини прямокутником. На рисунку 17 виділено вершини з 5 по 10.

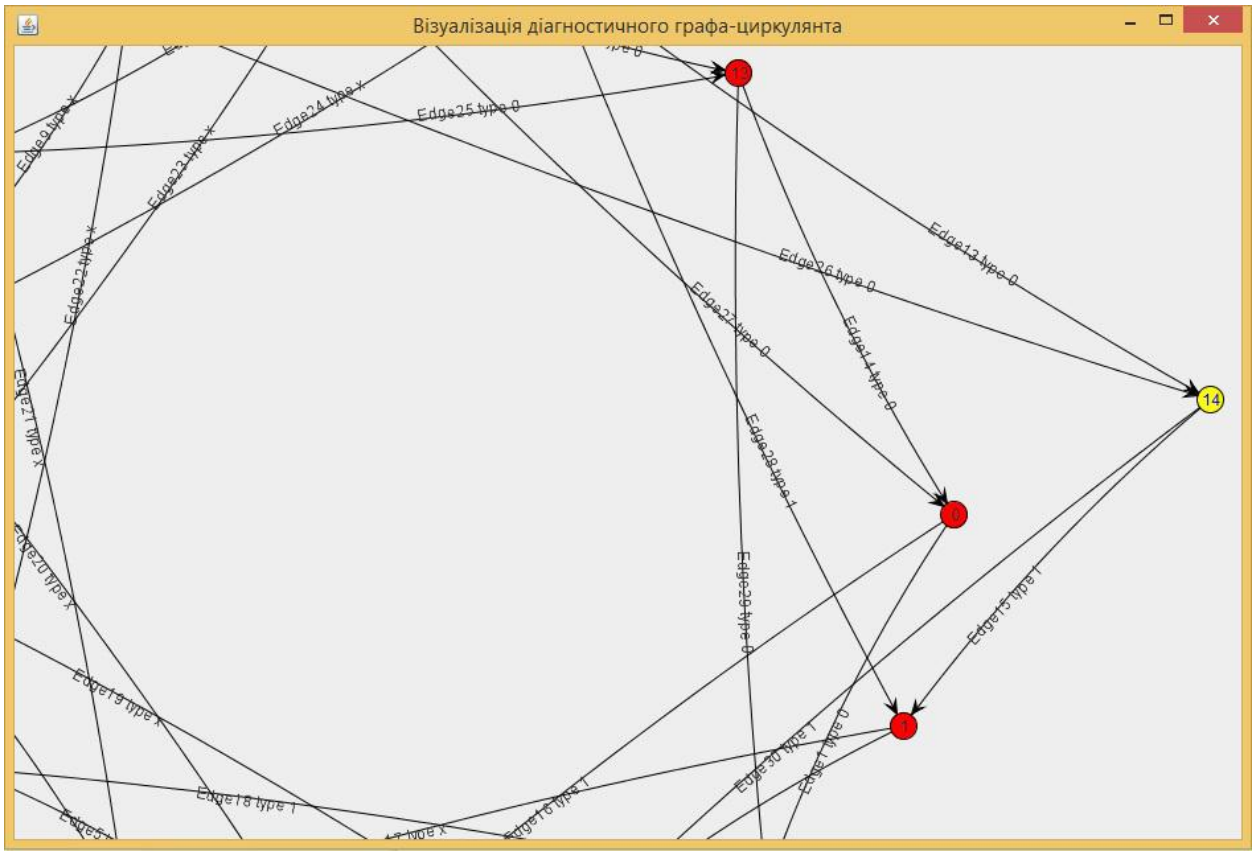


Рисунок 16 – Масштабування графа

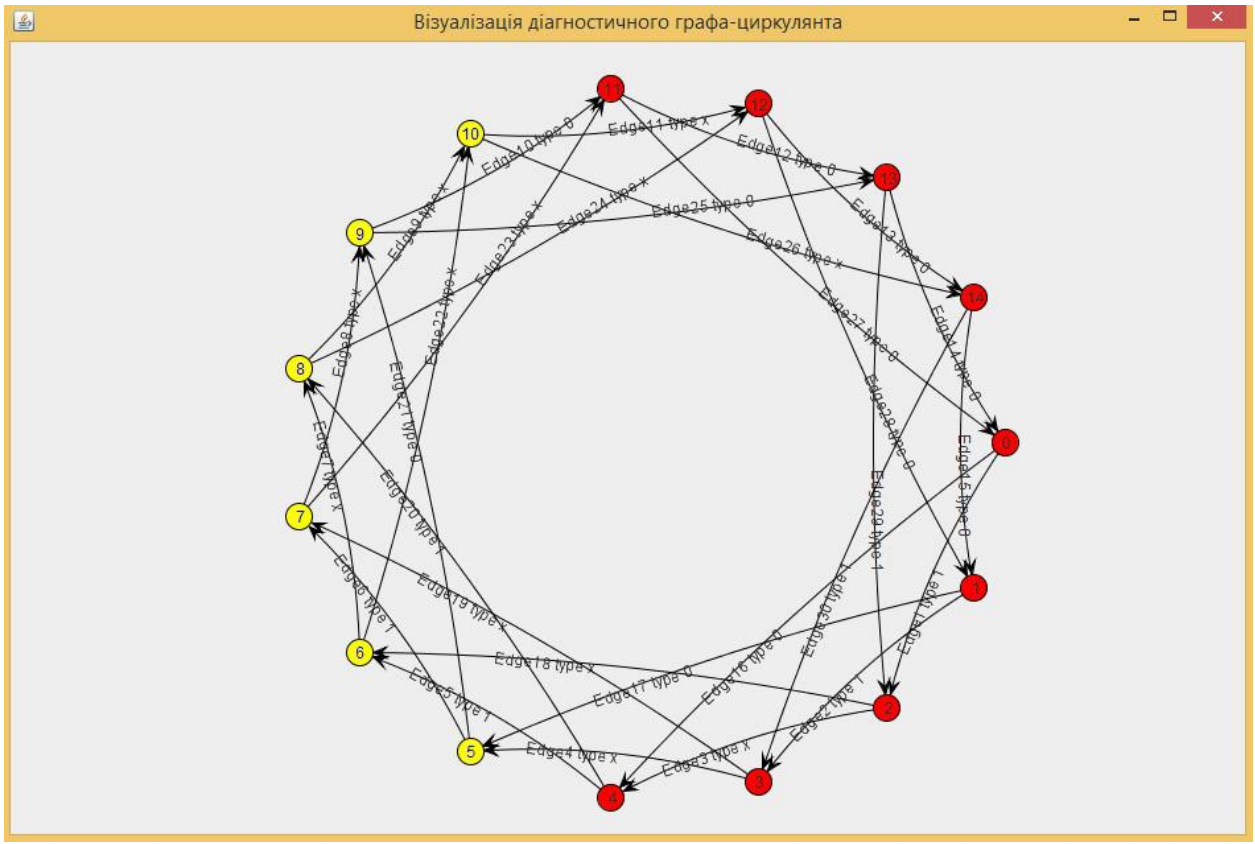


Рисунок 17 – Циркулянтний граф з 15 вершинами і стрибками цикла 2 і 4

Виділені вершини змінюють колір на жовтий. Потім мишкою, затиснувши будь-яку з виділених вершин, можна перемістити граф. При необхідності є можливість перемістити декілька виділених вершин, що зображено на рисунку 18

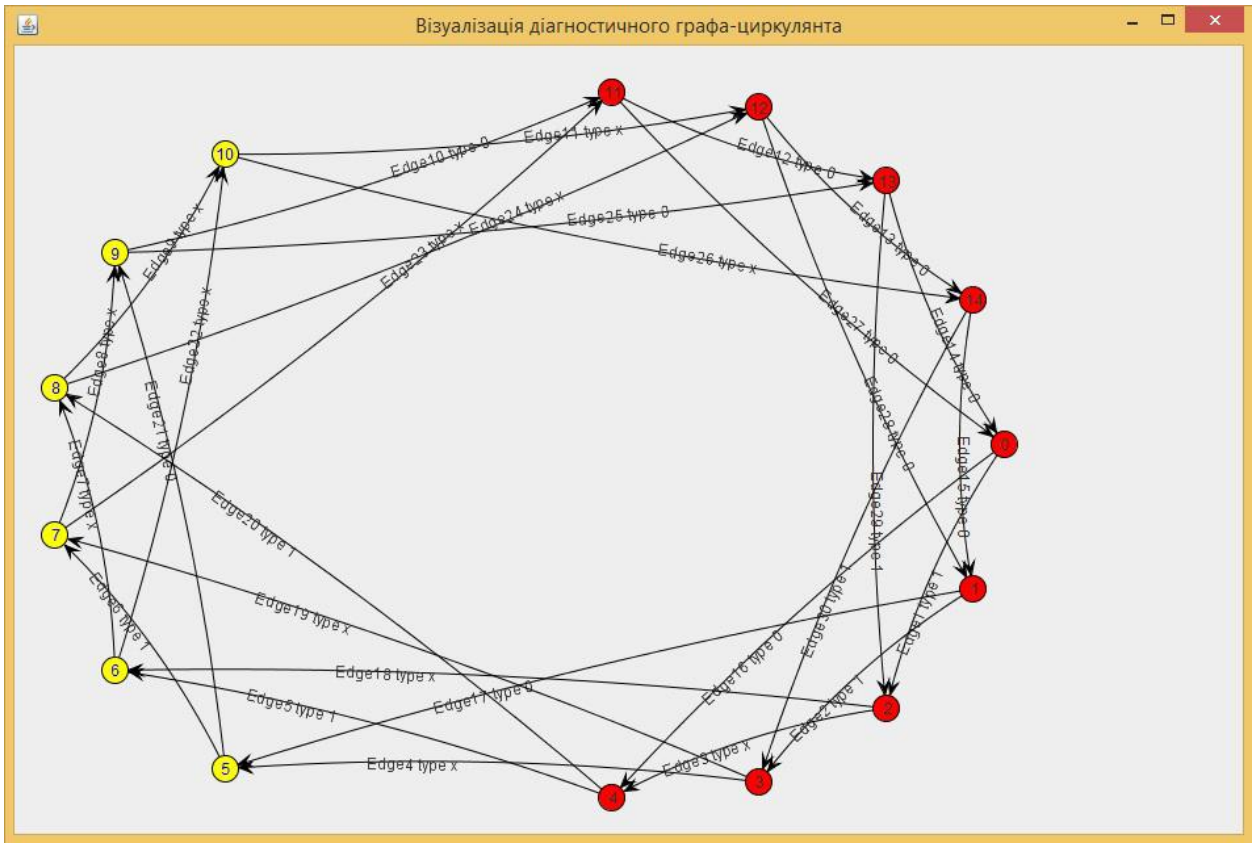


Рисунок 18 – Реалізація можливості переміщення вершин

3.2. Реалізація алгоритма побудови циркулянтного графа

Початковими даними є:

– amountOfVertex – кількість вершин;

– amounOfVertexTypeN – кількість вершин типу N;

– arrayOfCycles – вектор, що заповнюється значеннями стрибків циклів з вікна вхідних даних.

Спочатку створюється об’єкт типу DirectedSparseGraph і за допомогою addVertex та циклу від 0 до amountOfVertex будується кожна вершина.

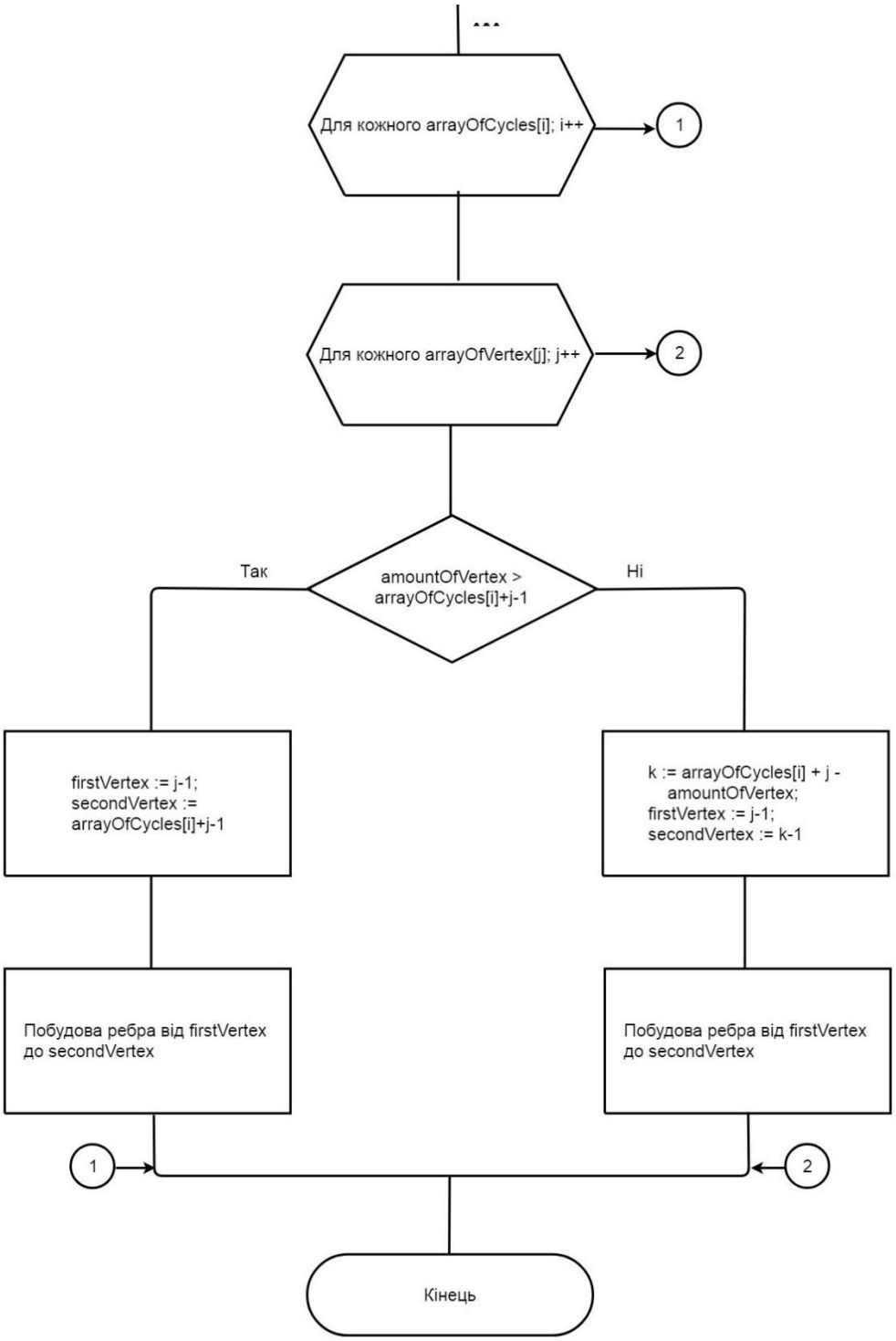


Рисунок 19 – Частина алгоритма побудови графа–циркулянта

Ініціалізується вектор arrayOfVertexTypeN. Циклом від 0 до amounOfVertexTypeN кожному елементу вектора arrayOfVertexTypeN присвоюється рамдомне число від 0 до amountOfVertex. Для виключення повторень чисел використовується метод numGen.

Ініціалізується змінна edge типу int для нумерації ребер і присвоюється їй значення “1”. Побудова графа реалізована за допомогою двох циклів та умовного оператора “if”, що показано на рисунку 19. В додатку 1 можна побачити повний алгоритм побудови графа–циркулянта.

3.3. Реалізація алгоритма формування ваги для кожного ребра

В модулі формування ваги реалізовано два метода:

– typeOfEdge(int v1, int v2);

– isExistInArray(int elem, int[] array).

Кількість вершин типу N задається користувачем. Вона не повинна перевищувати величину amountOfVertex/2.

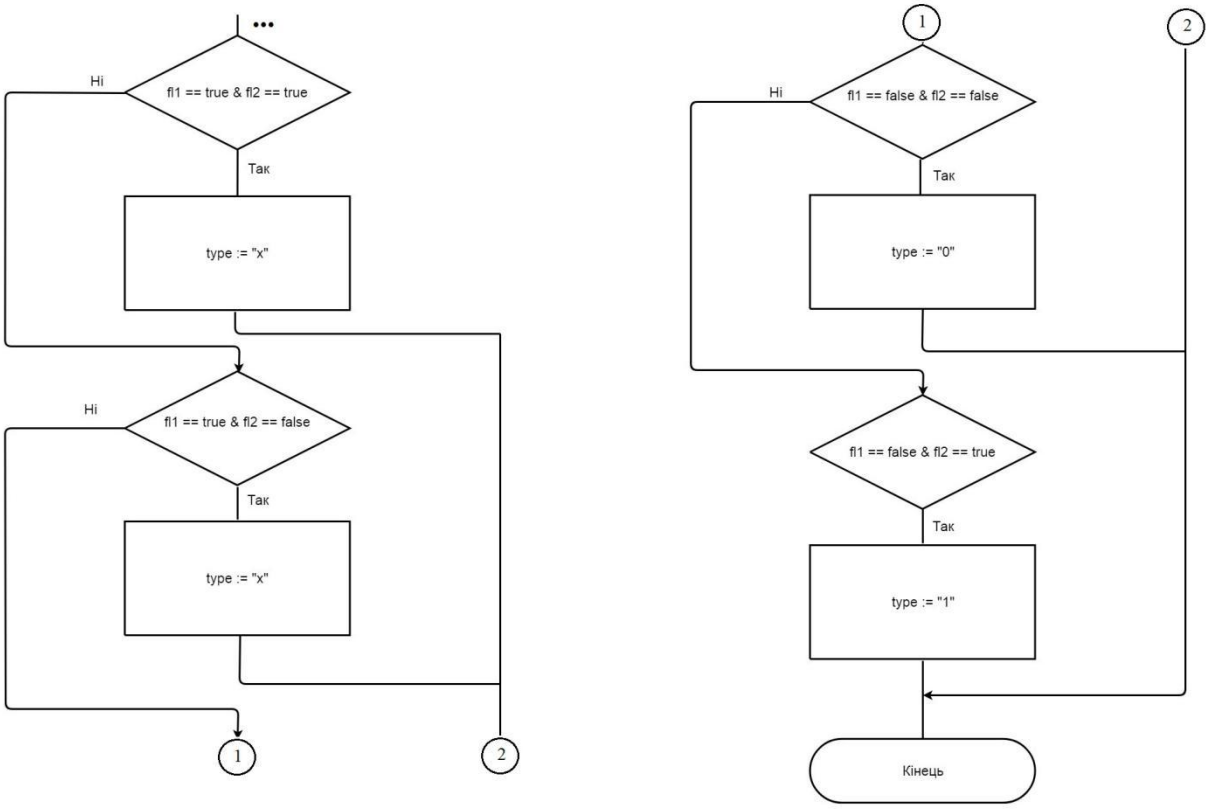


Рисунок 20 – Частина алгоритма формування ваги для кожного ребра

Метод isExistInArray визначає, чи міститься вершина у векторі вершин типу N. Він викликається з typeOfEdge для визначення типів двох вершин будь-якого ребра. В залежності від комбінації типів і за допомогою оператора “if” визначається, якою буде вага ребра.

Ці чотири випадки показано на рисунку 20. Повний алгоритм показано в додатку 1.

ВИСНОВКИ

Метою дипломного проекту було створення програми візуалізації циркулянтних графів для ілюстрації тестування в багатопроцесорних системах.

В процесі дослідження існуючих мов програмування було обрано одну з найбільш популярних мов програмування Java та бібліотеку JUNG для візуалізації елементів графів. Було розроблено алгоритми з урахуванням особливостей побудови графа-циркулянта.

В результаті аналізу існуючих рішень було виявлено ряд недоліків. Програми забезпечували або занадто загальний вигляд графів без врахування специфіки графа-циркулянта, або не мали можливостей зміни розташування графа. Більшість існуючих реалізацій мають можливість зберігати граф, але використовують незручний для користувача формат.

Вище перераховані недоліки були виправлені. Розроблена програма дозволяє:

– зображати діагностичний граф згідно зі вхідними даними;

– редагувати, масштабувати та переміщати створений граф;

– зберігати граф у форматі png.

Створений програмний продукт забезпечує полегшення і автоматизацію процесу діагностування, покращення засобів тестування. В перспективі може використовуватися як елемент реальної системи самодіагностування складних активно відмовостійких багатопроцесорних обчислювальних системах.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Di Battista G., Eades P., Tamassia R., Tollis I.G.Algorithms for drawinggraphs: an annotated bibliography // Comput. Geom. Theory Appl. – 1994. – Vol. 4. – P. 235–282.
2. Евстигнеев В.А., Касьянов В.Н.Толковый словарь по теории графов в ин–форматике и программированиию. – Новосибирск: Наука, 1999.
3. Sugiyama K., Misue K.Visualization of structured digraphs // IEEE Trans. onSystems, Man and Cybernatics. – 1991. – Vol. 21, N 4. – P. 876–892.
4. Шилдт, Герберт Java 8. Руководство для начинающих / Герберт Шилдт. – М.: Вильямс, 2015. – 620 с.
5. Бібліотека JUNG [Електронний ресурс] – http://jung.sourceforge.net/.
6. Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. – М.: Вильямс, 2015. – 459 с.
7. Романкевич В.А. Самотестирование многопроцессорных систем с регулярными диагностическими связями // Автомат. и телемех.– 2017, № 2.– С.115–127.
8. Romankevich V. A. Self–testing of multiprocessor systems with regular diagnostic connections // Automation and Remote Control. – 2017. – Vol. 78, Issue 2. – P. 289 – 299.
9. Романкевич А.М., Романкевич В.А. О диагностировании многопроцессорных систем при отказе более половины процессоров // Автомат. и телемех.– 2017, № 9.– С.84–90.
10. Эккель, Брюс Философия Java / Брюс Эккель. – М.: Питер, 2016. – 470 с.
11. Гупта, Арун Java EE 7. Основы / Арун Гупта. – М.: Вильямс, 2014. – 266 с.
12. Tutorial JUNG [Електронний ресурс] – http://www.grotto–networking.com/JUNG/JUNG2–Tutorial.pdf
13. Харари, Ф. Теория графов / Ф.Харари – М. : Мир, 1973. – 300 с.

14. Уилсон, Р. Введение в теорию графов / Р. Уилсон. – М. : Мир, 1977. – 207 с.
15. Belyavskii V.E., Valuiskii V.N., Romankevich A.M. and Romankevich V.A. Self–Diagnosable Multimodular Systems: Some Estimates of Testing // Autom. Remote Control. Y. 1999. V. 60. No. 8. P. 1179–1183.
16. Romankevich A. M., Romankevich V. A. Diagnosis of multiprocessor systems under failure of more than half processors // Automation and Remote Control. – 2017. – Vol. 78, Issue 9. – P. 1614 – 1618.
17. Романкевич В.А. Метод уменьшения количества взаимопроверок при самотестировании многопроцессорных систем / В.А. Романкевич, А.В. Романкевич, Д.Н. Ахмедова // Радіоелектронні і комп‘ютерні системи.– №4.– 2018.– С.61–66.
18. Романкевич В.О. Методи і засоби оцінки технічних характеристик гарантоздатності відмовостійких багатопроцесорних систем управління складними об’єктами: Дис. на здобуття наукового ступеня доктора технічних наук: 05.13.05; – Захищена 29.01.2018; Затв. 20.03.2018.– К., 2017.– 388с.
19. Моделирование. Тестирование, надёжность, контроль и диагностика компьютерных систем [Електронний ресурс]: учебное пособие для изучения дисциплин «Моделирование» и «Тестирование, надёжность, контроль и диагностика компьютерных систем» для иностранных студентов специальности «Компьютерные системы и сети», «Системное программирование» и «Специализированные компьютерные системы» / НТУУ «КПІ»; сост. В. В. Гроль, В. А. Романкевич, Е. Р. Потапова.– Електронні текстові дані (1 файл: 782.5 Кбайт). – Киев: НТУУ «КПИ», 2011.
20. Гроль В.В., Романкевич В.О. Базові поняття і конструкції мови програмування Сі. Методичні вказівки до вивчення дисципліни “Моделювання” // Київ.– “Політехніка”, 2003.– 24с.

21. Самофалов К.Г., Романкевич А.М., Валуйский В.Н., Каневский Ю. С. , Пиневич М.М. Прикладная теория цифровых автоматов. – К.: Вища Школа. – 1987г. – 375с.
22. Гроль В.В., Романкевич В.А., Потапова Е.Р. Организация процедур логического моделирования цифровых блоков. Методические указания к изучению дисциплин «Моделирование», «Тестирование, надёжность, контроль и диагностика компьютерных систем» (рос. мовою, для іноземних студентів) // Київ.– “Принт–центр”, 2007.– 44с.